Implementation of GARTO as an infiltration routine in a full hydrological model

Even Nyhus

Civil and Environmental Engineering
Submission date: June 2017
Supervisor: Knut Alfredsen, IBM

Norwegian University of Science and Technology
Department of Civil and Environmental Engineering

NTNU

NORGES TEKNISK NATURVITENSKAPLIGE UNIVERSITIET

Institutt for Vann- og miljøteknikk

Masteroppgåve i vassdragsteknikk

Kandidat: **Even Nyhus**

Tema: **Infiltrasjonsmetode i hydrologisk modell.**

1. Bakgrunn

Å kunne simulere infiltrasjon er viktig i mange samanhengar i hydrologi, og ikkje minst i samband med dimensjonering i småfelt eller felt der det skjer arealbruksendringar. I dei fordelte modellane som er vanleg i bruk i Norge har ikkje spesifikke infiltrasjonsmetodar, og det er difor ønskjeleg å få laga ein slik modell. Det er gjennom forprosjektet gjort ei grundig vurdering av moglege metoder for å implementere ei slik metode, og i denne oppgåva skal denne metoda implementerast i det hydrologiske rammeverket SHyFT (Statkraft Hydrological Forecasting Toolbox). Dette er eit fordelt modelleringssystem som er laga for at det skal enkelt kunne leggast til metodar og for å teste prototypar av nye hydrologiske verkty

2. Arbeidsoppgåver

Oppgåva vil ha følgjande hovuddelar:

1. Utvikle ei prototyp av metoda som er spesifisert i prosjektoppgåva. Denne skal vere frittståande og eigna for testing mot ulike oppsett. Køyre test av metoda mot litteraturdata og måledata for å sjekke funksjonalitet. Rutina skal dokumenterast både på kodenivå og for bruk.

2. Ein detaljspesifikasjon av den nye rutina må lagast med utgangspunkt i SHyFT sitt rammeverk. Dette gjeld algoritma, koplinga mot grensesnitt og naudsynte data og parametrar for rutina. Det er spesielt viktig å formulere arealbruksdata slik at dei passar med det eksisterande systemet.

3. Implementere metoda i SHyFT. Sjølve simuleringsalgoritma må kodast i C++ som ein del av simuleringsbiblioteket, medan grensesnittet må kodast i Python. Etter implementering skal testrutiner byggast etter SHyFT sin mal, og kode skal dokumenterast både med tanke på implementering, grensesnitt og krav til data og modellparameterar.
4. SHyFT skal no køyrast med den nye rutina implementert. Funksjon og resultat må sjekkast mot data frå 1) for å sjå om implementeringa fungerer om den skal.

5. Køyre testsimuleringar av den nye rutinar for eit par tilfelle:

a. Eit område med kjend infiltrasjon, t.d. eit grønt tak om slike data er tilgjengelege.
b. Det skal gjerast arealbrukssimuleringar for å systematisk vurdere korleis modellen reagerer på endringar i areal.

3. Rettleiing, data og informasjon

Faglærar vert professor Knut Alfredsen ved institutt for vann- og miljøteknikk, NTNU. Kandidaten er elles ansvarleg for innsamling, kontroll og bruk av data. Hjelp frå ovannemnde eller andre må refererast i rapporten.

4. Rapport

Struktur og oppsett av rapporten er viktig. Gå utifrå at det målgruppa er teknisk personell på seniornivå. Rapporten skal innehalde eit samandrag som gir lesaren informasjon om bakgrunn, framgangsmåte og hovudresultata. Rapporten skal ha innhaldsliste og referanseliste. Referanselista skal vere formatert etter ein eksisterande standard.

Denne oppgåveteksta skal vere inkludert i rapporten.

Data som er samla inn skal dokumenterast og leverast på digital form.

Formatet på rapporten skal følgje standarden ved NTNU. Alle figurar, kart og bilete som er inkludert i rapporten skal vere av god kvalitet med tydeleg tekst på akse og teiknforklaring.

Kandidaten skal inkludere ei signert fråsegn som seier at arbeidet som er presentert er eins eige, og at alle bidrag frå andre kjelder er identifiserte gjennom referanser eller på andre måtar.

Frist for innlevering er __. juni 2017.

Insitutt for vann og miljøteknikk, NTNU

Knut Alfredsen

Professor

Abstract

Climate change is expected to give more intense rainfall events, while urbanisation leads to more impervious surfaces. This combined with growing cities, increase the stress on the existing storm water infrastructure, and may result in more frequent flooding in urban areas.

The land use in urban catchments affects the stormwater runoff patterns. More impervious surfaces lead to runoff hydrographs reaching the flood peaks faster, and having higher peaks. Having green spaces and water retention on the other hand, results in hydrographs with a shape more similar to natural conditions: smaller peaks reaching maxima later.

To do urban hydrology assessments, e.g. for understanding the effects of land use changes, modelling of the hydrological processes in an urban area might be beneficial. Hydrological models can simulate the runoff patters from a catchment based on input parameters such as soil properties and rainfall patterns. Spatially distributed hydrological models are hydrological models that divides a catchment into smaller cells, each with its own properties and input parameters.

Spatially distributed hydrological models used in Norway today do not consider the mechanisms of infiltration and are therefore not applicable for urban hydrology assessments. This thesis therefore suggests including infiltration by implementing an infiltration routine in a hydrological model. The infiltration model Green-Ampt with redistribution Tablot and Ogden (GARTO) was chosen to create the base of the routine. Further, the widely used hydrological model Hydrologiska Byråns Vattenbalansavdelning (HBV) model was chosen for implementation of the infiltration routine. The HBV model already exists in open source database gihub, under Statkraft Hydrological Forecasting Tools (SHyFT).

The routine was coded separately in C++ and the results were compared to results from Lai et al. (2015). The routine was also tested against infiltration and soil moisture data of a green roof in Trondheim. The code was then implemented in SHyFT.

The comparison to the results from Lai et al. (2015) gave satisfactory results. The routine also managed to match parts of the results from a green roof. A sensitivity analysis on the soil properties shows that saturated conductivity is the most sensitive soil moisture constant. Further the routine match good with expected infiltration responses.

Sammendrag

Klimaendringene forventes å kraftigere nedbørshendelser. Samtidig fører urbanisering fører til flere tette flater. Dette kombinert med voksende byer, øker presset på den eksisterende overvannsnettet, og kan føre til hyppigere flom i byområder.

Areal bruk i byområdene påvirker flom avrenningen. Mer ugjennomtrengelige overflater gir en hydrograf med økte flomtopper som når maksimalverdier raskere. Å ha grønne områder og fordøyingsbassenger, resulterer i mer naturlig hydrograf med mindre topper som når maksima senere.

For å gjøre urbane hydrologiske vurderinger, f.eks. for å forstå effekten av endringer i arealbruk, kan modellering av hydrologiske prosesser i et byområde være en god løsning. Hydrologiske modeller kan simulere avrenningen fra et nedbørsområde basert på inngangsparametere som jordartsegenskaper og nedbør. Fordelte hydrologiske modeller er hydrologiske modeller som deler et nedbørsfelt i mindre celler, hver med sine egne egenskaper og inngangsparametere.

Fordelte hydrologiske modeller som brukes i Norge i dag, inneholder ikke egne infiltrasjons rutiner og er derfor lite anvendelige for urbane områder. Denne masteroppgaven foreslår derfor å inkludere infiltrasjon ved å implementere en infiltrasjonsrutine i en hydrologisk modell. Infiltrasjonsmodellen Green-Ampt with redistribution, Tablot and Ogden (GARTO) ble valgt som utgangspunkt for rutinen. Videre ble det valgt å implementere infiltrasjonsrutinen i den populære hydrologiske modellen Hydrologiska Byråns Vattenbalansavdeling (HBV). HBV-modellen eksisterer allerede i opensource databasen gihub, under Statkraft Hydrological Forecasting Tools (SHyFT).

Rutinen ble kodet separat i C ++, og resultatene ble sammenlignet med resultater fra Lai et al. (2015). Rutinen ble også testet mot infiltrasjon og jordfuktighetsdata fra et grønt tak i Trondheim. Koden ble deretter implementert i SHyFT.

Sammenligningen med resultatene fra Lai et al. (2015) ga tilfredsstillende resultater. Rutinen klarte også å matche deler av resultatene fra det grønne taket. En sensitivitetsanalyse på jordartsegenskaper viser at mettet hydraulisk konduktivitet er den mest følsomme jordartsparameteren. Videre samsvarer rutinen godt med forventede infiltrasjonsresponser.

Preface

This thesis is submitted to the Norwegian University of Science and Technology (NTNU). It is a product of the course *TVM4910 Hydropower engineering, Master's Thesis*. The thesis describes the development and implementation of an infiltration routine in a full hydrological model.

The study was conducted at the Department of Civil and Environmental Engineering. I would like to express my gratitude to my supervisor Professor Knut Alfredsen. Thank you for good advices regarding infiltration processes and hydrological models. I would also like to thank Knut Alfredsen, Sigbjørn Helset and Kuganesan Sivasubramaniam for the help with implementing the infiltration routine in SHyFT. Alfredsen and Sivasubramaniam have helped me understanding how implement the infiltration routine, while Helset has supported me by correcting debugging errors as well as giving me helpful tips on how to build the routine in SHyFT.

I would also like to thank:

- Birgitte Gisvold Johannessen for providing infiltration data from the green roof
- Guro Heimstad Kleiven for correcting of both technical issues and spelling errors
- Jenny Kahrs Henriksen for spelling errors

The work in this thesis is done by the Author, Even Nyhus. References are included on all collected data and citations that are not the work of the author.

Trondheim, 2017

Even Nyhus

Even Nyhus

Table of Contents

List of figures

List of tables

List of Acronyms

Physical description	**Acronym**	**Symbol in the code**	**Unit**
Precipitation	*Pe*	*Pe*	[LT^{-1}]
Green Ampt wetting front soil suction	*Hc*	*Hc*	[L]
Bubbling pressure/air entry pressure	ψ_b	*Yb*	[L]
Pore distribution index	*Λ*	*lambda*	[-]
Saturated volumetric soil moisture content	θ_s	*Os*	[-]
Residual volumetric soil moisture content	θ_r	*Or*	[-]
Initial volumetric soil moisture content	θ_i	*Oi*	[-]
Saturated hydraulic conductivity	K_s	*Ks*	[LT^{-1}]
Initial hydraulic conductivity	K_i	*Ki*	[LT^{-1}]
Relative volumetric initial water content	Θ_i	*ORi*	[-]
Relative volumetric water content	Θ	*OR0*	[-]
Number of wetting fronts	*K*	*k*	[#]
Current hydraulic conductivity	$K(\theta)$	*K0*	[LT^{-1}]
Current volumetric soil moisture content	θ	*O0*	[-]
Infiltrated physical depth	*Z*	*Z*	[L]
Actual infiltration	*f*	*f*	[LT^{-1}]
Infiltrated water depth	*F*	*Freal*	[L]
Water demand for wetting front 0-(k-1)		*Pek*	[L]
Residual rain water		*PeR*	[L]
Porewater volume for wetting front 0-(k-1)		*Vzk*	[L]
Unsaturated capillary drive	*G*	*G*	[L]
Infiltration rate if *Pe* > *f* since the start		*ftest*	[LT^{-1}]
Infiltrated water before the start		*deltaF*	[L]
Rainfall rate in last timestep		*PeL*	[L]
Soil water content during iteration		*O0test*	[-]
Wetting front depth during iteration		*Ztest*	[L]
Infiltrated water during iteration		*Ftest*	[L]
Traditionally Green-Ampt infiltration model	GA		
Richards Equation	RE		
Green-Ampt with Redistribution	GAR		
Modified GAR	MGAR		
Tablot and Ogdens infiltration model	TO		
Combination of GAR and TO	GARTO		
GA with Mein Larson modification	GAML		

1 Introduction

Rainwater that hits the ground either evaporates, transpires, infiltrates or creates surface runoff (Horton, 1933). Urbanisation as we know it today has been shown to shift the allocation of water between these three processes. More of the rainwater creates runoff, due to higher density of impervious surfaces. On top of this, rainfall amounts and intensities are expected to increase in Norway in the future, due to climate change (NKSS, 2015).

Urban stormwater management is therefore of increasing concern in the context of urban planning. To avoid flooding, the infrastructure needs to be designed for future climate. Hydrological models can be used for forecasting the need for stormwater capacity in urban drainage systems, or to anticipate the effects of land use changes.

Distributed hydrological models divide the catchment into smaller cells, each with its own input parameters and properties. The majority of the spatially distributed hydrological models are developed for natural watersheds. Simulating infiltration is important in hydrology, and it becomes even more important in urban areas, due to the varying perviousness of the land cover and the complexity of the urban soils. The already-existing spatially distributed hydrological models do not have an infiltration routine and are therefore not applicable in urban watersheds. (Salvadore et al., 2015).

Including an infiltration routine could enhance the hydrological models and make them more applicable for handling the complexity of urban watersheds. The aim of this thesis is therefore to include infiltration by developing an infiltration routine and implementing it in a full hydrological model.

Finding the right infiltration method for implementation in a hydrological model is important. A thorough analysis of possible infiltration methods was conducted in *TVM4520 Specialization project*, fall 2016. The method Green-Ampt with redistribution (GAR) (Ogden and Saghafian, 1997) was found best suited. In this thesis, an infiltration routine based on GAR is coded in C++ before it is implemented.

Theory

2 Urban surface water and infiltration

Urbanized areas are constantly growing. In 2009, more than half the world's population lived in urban areas, and an increase to more than 80% by 2030 is expected. The rainfall amounts and intensities are expected to increase in Norway in the future due to climate change (NKSS, 2015) and urban areas have been found vulnerable to heavy rainfall (Salvadore et al., 2015) and

In nature, a substantial portion of the rainfall that hits the ground is infiltrated into the ground or evaporated as shown in Figure 2.1.a. The remaining water becomes surface runoff. Both groundwater and surface runoff drains to the nearest river or reservoir. Groundwater transport is a slow process that retains the water. Surface runoff is a faster process. Urbanisation as we know it today involves piping of rivers, removal of vegetation, modification of natural soil and large areas being covered by impervious surfaces. This changes the allocation of water to the evapotranspiration, infiltration, and surface runoff. The consequence is less water returned to the atmosphere and infiltrated to the ground and more water occurring as surface runoff as shown in Figure 2.1.b. In addition to larger amounts of runoff, will the velocity of the runoff increase due to less friction in pipes and on surfaces, compared to natural conditions.

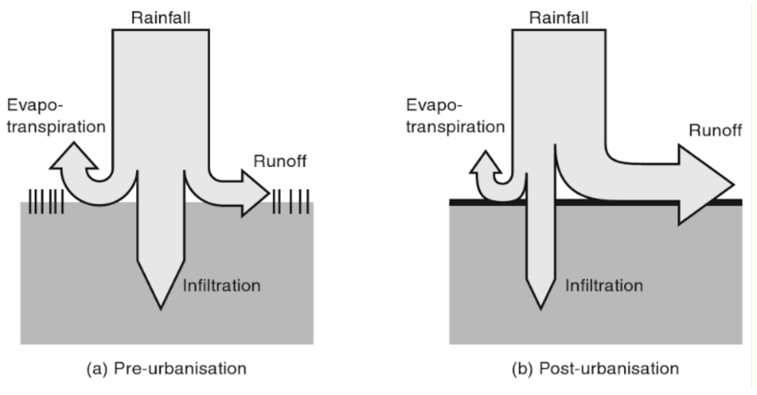


Figure 2.1 *The effect of urbanisation on the stormwater runoff (Butler and Davis, 2004)*

The natural hydrograph will have a low peak and a large lag time from the precipitation event starts until the runoff peak is reached. The urbanized hydrograph, on the contrary, will have a high flow peak which will come fast (Butler and Davis, 2004).

As an example of this, it was found in a study in Korea that for a specific rainfall event, the hydrograph peak is 15% higher than before urbanization. The lag time (Figure 2.2) was found to be reduced with 70 minutes (Kang et al., 1998). The combination of more water passing a point in a shorter time interval, results in a greater maximum flow (Butler and Davis, 2004).

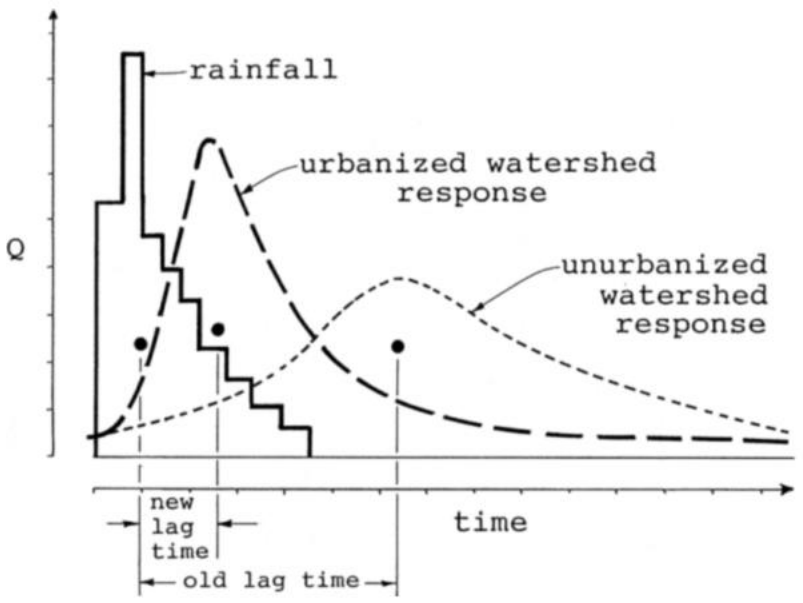


Figure 2.2 Illustration of the effect of urbanisation on a hydrograph (Rodgers and Hasselmann, 1997)

The traditional way of handling stormwater and wastewater with piped systems can be referred to as “hard engineering solutions” (Butler and Davis, 2004). However, the combination of expanding cities and increased precipitation leads to overloading of the wastewater system, which again leads to flood events causing great harm to cities. A new philosophy of “sustainability” has therefore grown during the last years within stormwater management. The system should not depend as much on “hard engineering solutions”, but recover some of the natural mechanisms by reopening rivers and creating green spaces (Butler and Davis, 2004). Water is again allowed to infiltrate, and to move as groundwater to the rivers. This way the hydrograph is stretched out and the peak runoff is decreased. The goal is to move from situation (b) back to situation (a) (Figure 2.1) by utilizing the infiltration capacity of the soil and retaining water to avoid the highest hydrograph peaks.

3 Infiltration

The key solution in sustainable urban planning regarding stormwater management is to utilize the soil's ability to infiltrate and retain water. Therefore, mechanisms of infiltration will be described in the following sections.

Darcy found a relationship describing flow through a porous medium:

$$Q = -KA\frac{dh}{dL} \tag{1}$$

He discovered that infiltrated water, Q [L^3T^{-1}], is proportional to the cross section A [L^2] and the head loss dh [L] and inversely proportional to the flow length dL (m). K [LT^{-1}] is the hydraulic conductivity constant which describes the rate the water can flow through a porous medium (Dingman, 2015). Modifications of Darcy's law is the basis for several of the infiltration models described later.

3.1.1 Infiltration during a rainfall event

Two important terms when working with infiltration of water into soils are *infiltration capacity* and *actual infiltration rate*. There is a slight difference between these two terms. Infiltration capacity, denoted in this thesis as f_c [LT^{-1}], is the *maximum* speed in which the water can enter the soil at a certain state. Actual infiltration rate, denoted as (f) [LT^{-1}] in this thesis, describes the speed of the which water is entering the soil at a certain state. f equals f_c when the rainfall intensity equals or is larger than f_c. Otherwise the f equals the rainfall rate. .f_c is affected by the soil moisture content and the capillary drive (G). G is the integrated capillary head across the wetting front (Horton, 1933).

The hydraulic conductivity K_s of a soil will vary, depending on the soil moisture content. When the soil is fully saturated, the term *saturated* hydraulic conductivity is used. This is the maximum hydraulic conductivity a soil can have, and it is dependent on the soil type. Any lower soil moisture content will result in a lower hydraulic conductivity (Lai et al., 2015).

Infiltration occurs if there is available surface water. Surface water can occur as ponded water, or rainfall water. Situations A-D shown in Figure 3.1 describe the infiltration rate depending on infiltration capacity and rainfall intensities (Mein and Larson, 1973).

In situation (A), the rainfall intensity is less than the saturated hydraulic conductivity. All the rain water will infiltrate into the soil and no ponding will occur. The hydraulic conductivity of the soil will be greater than the rainfall intensity so the soil moisture content will decrease.

In situation (B), the rainfall intensity is greater than the hydraulic conductivity, but lower than f_c. In this situation too, all the rain water will infiltrate. In situation B, the hydraulic conductivity of the soil will be lower than the rainfall intensity, so the soil moisture content of the upper layer of the soil will gradually increase until it reaches full saturation.

When the soil reaches full saturation and the same rainfall intensity continues, situation (C) will occur. Because f_c has been decreasing during situation B, the rainfall intensity is now greater than f_c, and the upper layers of the soil become saturated. Ponding will occur.

If the rainfall intensity is greater than f_c at the beginning of a rainfall event, Situation (D) will be present. f will in this case equal f_c during the rainfall event, or until the rainfall intensity drops below f_c.

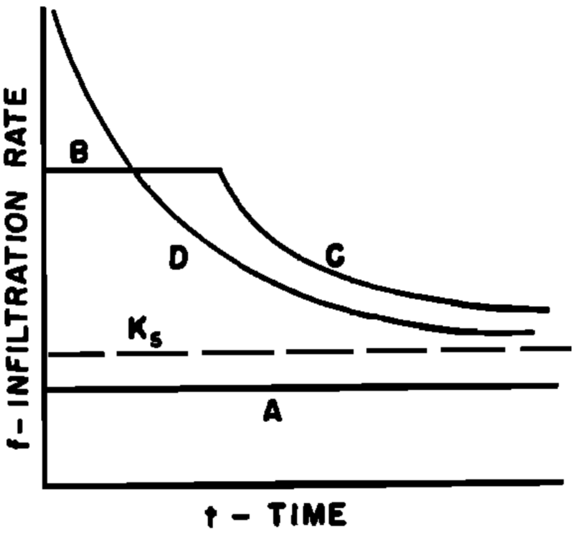


Figure 3.1 *Infiltration rate for situation A,B,C and D (Mein and Larson, 1973)*

Different infiltration situations are now explained. The infiltration process is, however, complex. The conditions impacting the infiltration capacity can differ severely over small distances. Layers in the soil column, types of soil, vegetation and the soil moisture distribution affects the infiltration capacity of the soil. Even a soil profile with several layers and with an evenly distributed soil moisture content can be complex to calculate (Mein and Larson, 1973).

3.1.2 Soil moisture redistribution

After a rainfall event, when all the surface water has infiltrated, the soil moisture will start to move downwards in the soil column, due to gravitational and capillary forces. This process is called redistribution (Ogden and Saghafian, 1997). Figure 3.2 presents the shape and location of a wetting front through a soil column at two different points in time. The total water volume in a soil column at a given time is the area of the graph multiplied with the surface area of the soil column.

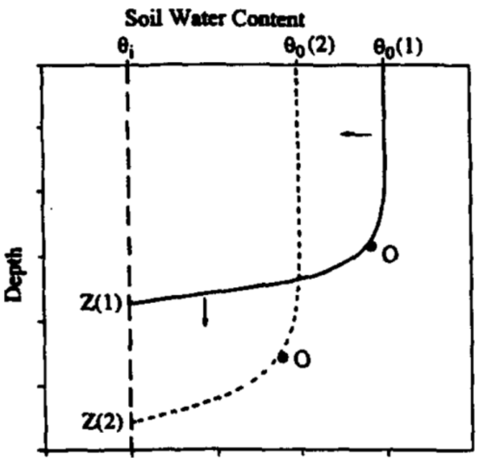


Figure 3.2 *Redistribution of water (Corradini et al., 1997)*

The y-axis represents the depth (Z) [L] and the x-axis represent the soil moisture content (θ_0). If no water is added to the soil column and evapotranspiration is neglected, the total water volume will be constant. Over time, the soil moisture will redistribute form situation (1) to situation (2) in Figure 3.2. When the wetting front percolates to an increased depth form Z(1) to Z(2), the soil moisture content will decrease from $\theta_0(1)$ to $\theta_0(2)$ to maintain mass balance (Ogden and Saghafian, 1997).

Many studies have described the infiltration process through saturated soil, but few studies have been conducted on unsaturated soil and the redistribution process (Bunsri et al., 2008). The following methods for calculation of infiltration capacity do all consider redistribution.

3.2 Infiltration calculation methods

In the *specialisation project* mentioned in section 1, several infiltration methods were evaluated. The GAR method was found to be the best fitted method for describing infiltration in a full hydrological model. In the following sections, Richards Equation and several modifications of the GAR method are presented.

3.2.1 Richards Equation

Richards Equation (RE) is an infiltration model with steady linear flow and a 1-D system that can be expressed as:

$$K\frac{d^2\psi}{dz^2} + \frac{dK}{dz}\frac{d\psi}{dz} + g\frac{dK}{dz} = 0 \tag{2}$$

where K is the hydraulic conductivity, z is the depth of the soil, $g = \frac{d\theta}{dz}$ is the change in soil moisture over the depth z, and ψ [L] is the capillary drive. RE is developed from Darcey's law Equation 1, but Richard also includes capillary drive and gravity forces (Richards, 1931).

RE is commonly accepted as the most accurate description of infiltration and is often used as a reference for development of new infiltration models (Ross, 1990). There are several proposals for solving RE. Flux-Updating Iterative Conjugate Gradient (FUCG) is an algorithm for solving the RE for saturated soil, based on the two-dimensional pressure-based form of Richards' Equation (3).

$$C\frac{dh}{dt} = \frac{d}{dy}\left(K - K\frac{dh}{dy}\right) - \frac{d}{dx}\left(K\frac{dh}{dx}\right) \tag{3}$$

C is $\frac{d\theta}{dh}$ and K is the saturated hydraulic conductivity. FUCG is able to calculate soil moisture content in a two-dimensional system and performs with high accuracy (Kirkland et al., 1992).

Soil hydraulics are highly nonlinear, which causes the RE to be computational expensive. RE also struggle with some conditions like overly dry soil, near saturation and nonuniform porous media (Lai et al., 2015).

3.2.2 Green-Ampt with Redistribution (GAR)

The Green-Ampt with redistribution (GAR) is an infiltration model developed by Ogden an Saghafian (1997) which describes the redistribution of soil moisture content θ during a rainfall hiatus. GAR uses two wetting fronts that develop separately to describe the soil moisture situation in a soil column. GAR uses a redistribution equation, Equation 7, based on the model of Smith et al (1993) and combines it with the Green-Ampt infiltration model (Green and Ampt, 1911). The Green-Ampt model (GA) calculates the infiltrated water depth and the infiltration capacity during heavy rainfall with initial soil moisture content and infiltrated water depth as input values, as shown in Equation 4. During a hiatus, Smith et al. (1993) continuously calculates the change in θ which again is used as input values by GA for the next rainfall event (Ogden and Saghafian, 1997):

$$f(t) = \frac{dF(t)}{dt} = K_s\left[\frac{(\theta_s - \theta_i)Hc}{F(t)} + 1\right] \tag{4}$$

Where *f(t)* is the infiltration capacity, *F(t)* is the infiltrated depth, K_s is the saturated conductivity, θ_s and θ_i is saturated and initial soil moisture content respectively and H_c is the Green-Ampt soil suction parameter. Lai et al. (2015) tested GAR against FUCG RE and the results were satisfactory. However, for coarse soils, the evolution of surface soil water moisture content in subsequent long periods of redistribution was not accurately predicted (Lai et al., 2015). The Modified GAR (MGAR) model is an enhancement of GAR which uses multiple wetting fronts. MGAR also introduces a coefficient. MGAR reduces the error in surface water content predictions (Gowdish and Muñoz-Carpena, 2009). However, neither GAR or MGAR can consider shallow water table conditions, which limits the model to catchments with deep unsaturated soil profiles.

3.2.3 Talbot-Ogden (T-O)

The Talbot-Ogden (T-O) model describes an infiltration model where the redistribution equation, Equation 7, in GAR is calculated with respect to depth Z and not with respect to soil moisture content θ, as in the GAR and MGAR models. The model also divides the soil profile into several bins with a size of $\Delta\theta$ as shown in Figure 3.3.

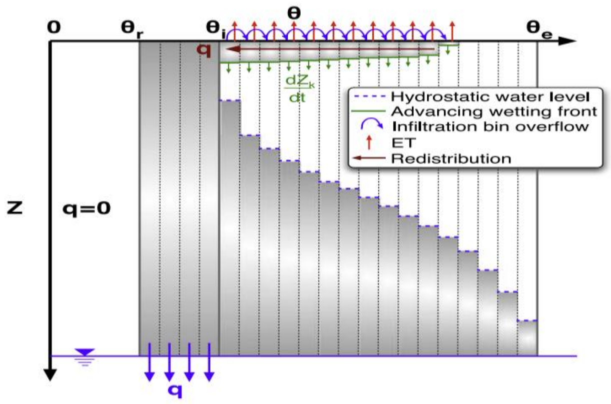


Figure 3.3 *Schematic illustration of the T-O model (Talbot and Ogden, 2008)*

The T-O model considers both the advance of a wetting front starting at the soil surface and the development of a water table further down in the soil profile. Each bin therefore considers two

Z-values. One Z-value describes the depth of the wetting front and one Z-value describes the depth of the water table for this bin. When the wetting front reaches the water table, the bin is saturated throughout the soil column like for the first four bins in Figure 3.3 (Talbot and Ogden, 2008).

3.2.4 GARTO

GARTO is a model that combines the GAR model and the T-O model. The model is efficient and guaranteed stable (Lai et al., 2015). Like the T-O method, GARTO controls several wetting fronts and can also control a groundwater table. Each wetting front is controlled by defining θ_i, θ and Z of the wetting fronts. θ_i coincides with the θ-value of the last wetting front. The figure below illustrates the differences between GA, GAR and GARTO.

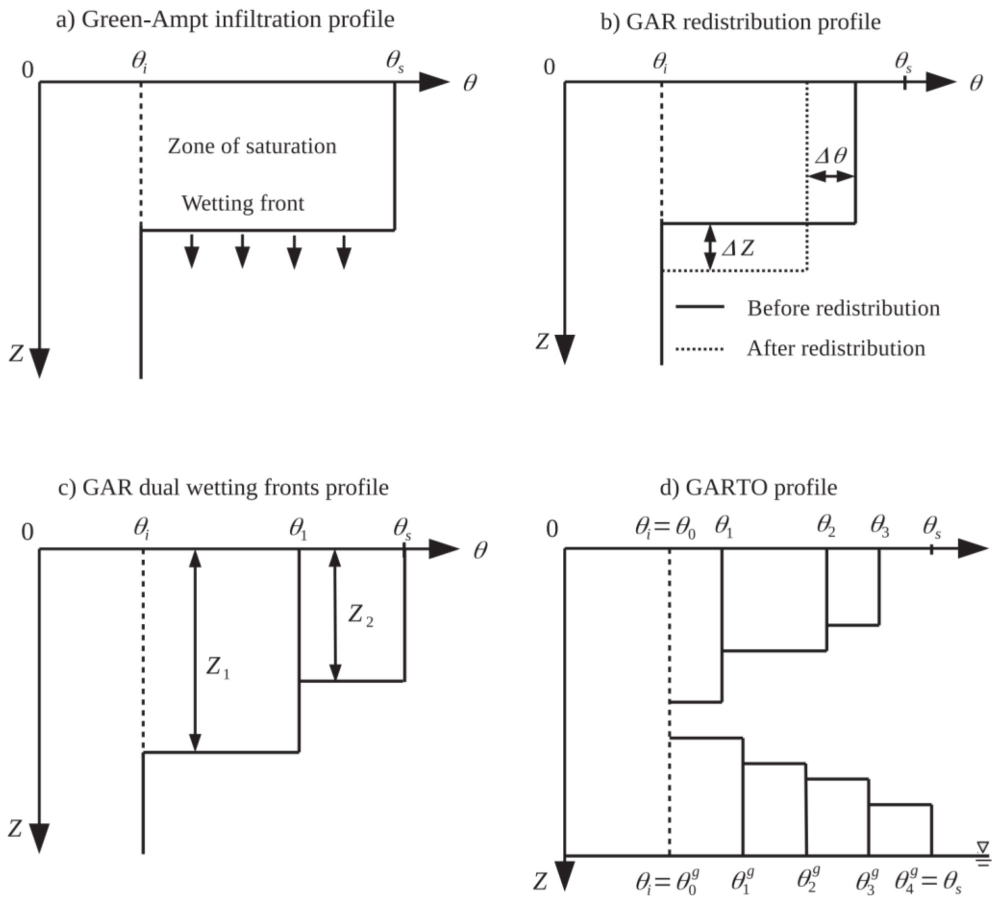


Figure 3.4 *Comparison of the GA, GAR and GARTO methods (Lai et al., 2015)*

GARTO uses GA under ponded conditions to calculate total infiltration depth (F) and f_c. For the last wetting front, under ponded conditions, θ equals θ_s.

$$F(t) = K_s t + H_c \Delta\theta \ln\left[1 + \frac{F(t)}{H_c \Delta\theta}\right] \tag{5}$$

$$f_c(t) = K_s \left[1 + \frac{H_c \Delta\theta}{F(t)}\right] \tag{6}$$

where K_s is saturated hydraulic conductivity [LT^{-1}], t is time [h], H_c is the wetting front suction head [L], $\Delta\theta = \theta_s - \theta_i$ where θ_s and θ_i is the saturated and initial volumetric soil moisture content respectively [-]. The redistribution equation, Equation 7, is used between precipitation events to control the developing soil moisture content:

$$\frac{\Delta\theta}{dt} = \frac{1}{\beta Z}\left[r - K(\theta) - \frac{pK_s G(\theta_i,\ \theta_0)}{Z}\right] \tag{7}$$

where r is the rainfall intensity, $K(\theta)$ is the conductivity at current soil moisture content [LT^{-1}], β is a shape constant [-] set to 1, p is a constant [-] set to 1,0 when $r > 0$ and 1,7 when $r = 0$. By controlling the soil moisture content (θ), GARTO applies the θ-value of the last wetting front as θ_i in GA for a new rainfall event.

For the first timestep in a new rainfall event, a maximum advance during a timestep or "dry depth" is approximated as:

$$h_{dry} = 0{,}5\left(\sqrt{\tau^2 + 4\tau G(\theta_i, \theta_s)}\right): \qquad \tau = \frac{\Delta t K_s}{\theta_s - \theta_i} \tag{8}$$

where h_{dry} is the first Z-value for the new wetting front and G is the capillary drive. Variations of Equation 12 and 13 calculate further advance of the wetting fronts as explained later.

The soil moisture content θ changes only in the last wetting front during redistribution. The other wetting fronts keep a constant θ. Water needed to continue the advance in Z direction is taken from ponded water, or from the last wetting front if there is no ponded water.

If the depth of one wetting front exceeds the depth of the adjacent wetting front to the left, the wetting fronts are merged together by the given equation:

$$Z_{k-1}^{new} = \frac{(\theta_k, \theta_{k-1})Z_k + (\theta_{k-1} - \theta_{k-2})Z_{k-1}}{(\theta_k - \theta_{k-2})}: \qquad \theta_{k-1}^{new} = \theta_k \tag{9}$$

Where k is the number of wetting fronts considered counting from the left in Figure 3.4.d.

GARTO provides a numerical stable method that it is proven to match well with the RE method in several situations. It also allows for continuous calculations of infiltration capacity by evaluating redistribution. GARTO is therefore chosen to establish a base method for an infiltration routine in a full hydrological model.

4 Hydrological models

When designing stormwater managing infrastructure, hydrological models are potentially powerful tools. they can e.g. predict how changes in a watershed will affect the runoff. The impact of future urbanisation, climate change and changing rainfall patterns can all be simulated by hydrological models and be used as design criteria for new water infrastructure (Jacobson, 2011). However, the application in urban areas are limited today as few models suited for urban assessments exist. This thesis suggests implementing GARTO as an infiltration routine in a full hydrological model. This will make the model better equipped to handle stormwater predictions in urban areas.

To set a suitable scope for the hydrological model is important. Hydrological models are developed and used within weather forecasting, irrigation modelling, hydropower economical assessments, water quality simulations, flood protection and other fields. The scope for the models within these fields varies. A large scope with a need of detailed data mapping is more computational expensive to run. Weather forecast models are example on models with a large scope. Choosing a scope that provides accurate enough results on a reasonable amount of computational power is important. “Watershed hydrology” is a term that deals with hydrological processes on a watershed scale to determine the watershed response (Singh, 1995). A watershed provides accurate boundaries for input values and responses. A watershed is also a suitable size for hydrological models and is therefore a suitable scope for hydrological models simulating urban stormwater.

Hydrological models can be categorized into three groups: physically-based, conceptual, and empirical models. Empirical models analyse observed input and output to obtain a statistical relationship between them. This requires only simple mathematical computations and provides fast results. Empirical models will however not provide very accurate results. Physically-based models consider mass balance equations of water through the watershed. This requires high computational power and detailed knowledge of the watershed parameters. If the input data is good, these models provide accurate results. Conceptual models describe hydrological processes and uses simple mathematical equations to describe the mass balance in each process e.g. evapotranspiration, snowmelt, surface runoff, surface storage and percolation. Parameters baked into these equations are used to calibrate conceptual models by using historical data. Hydrological models are also divided into distributed or lumped models. The difference is whether the model divides the watershed in smaller subbasins or considers the watershed as one single unit (Aghakouchak and Habib, 2010).

Around the world, several models describe the movement of water at watershed level. The HBV-model is the standard model for flow forecasting in Scandinavian countries (Singh, 1995). It will therefore be used as a reference model for implementation of an infiltration routine. The principal of an infiltration routine could however be implemented in several hydrological models.

4.1 The HBV model

The HBV model was originally developed as a lumped conceptual model, but was modified to a distributed model using subbasin division in 1996. The model setup is simple and the data demand is low. Nevertheless, the model is robust and gives precise results, e.g. demonstrated by a study where the model returned results with the coefficient of determination (R^2) around 89%, for seven different test areas (Lindström et al., 1997). The HBV-model is a continuous model, meaning that it can simulate several rainfall events following each other. The time series with data are divided into incremental timesteps.

A schematic setup for a modified HBV model is shown in Figure 4.1. This setup was used in a study researching the effect of urbanisation and more impervious surfaces in a subbasin of Rhine (Bunsri et al., 2008). The model takes three input parameters and runs them through some "boxes" simulating hydrological processes. The input parameters are precipitation, air temperature and potential evaporation. Each "box" is called a *routine* and contains mathematical equations that describe a specific hydrological process. Each routine takes input and generate output values. The output values are calculated inside the routine. All the routines, except impervious surfaces, can also store water. The output from the snow routine becomes the input in the soil moisture routine and so on. For each new timestep, an algorithm runs through all the routines and produces a subbasin response, Q [L^3T^{-1}]. Q is the total runoff generated from lower reservoir, upper reservoir, and surface runoff in Figure 4.1. If the model is distributed, the algorithm will run through all the subbasins with respective input values and produce a Q for each subbasin.

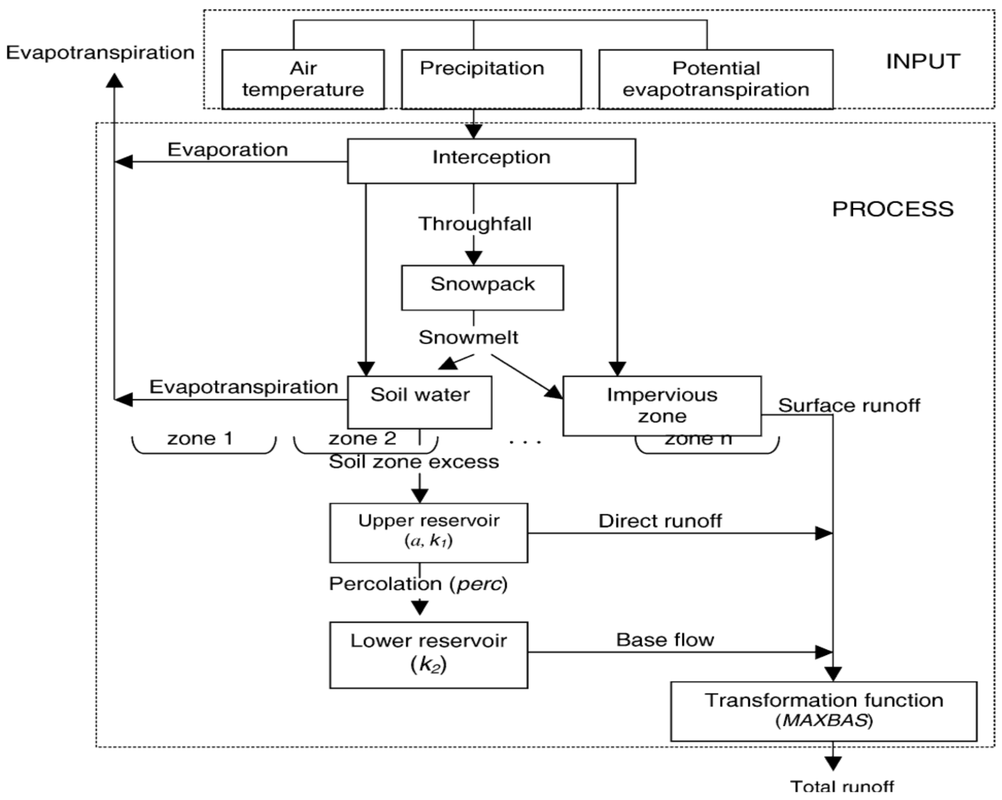


Figure 4.1 *Schematic setup of the HBV-model (Hundecha and Bárdossy, 2004)*

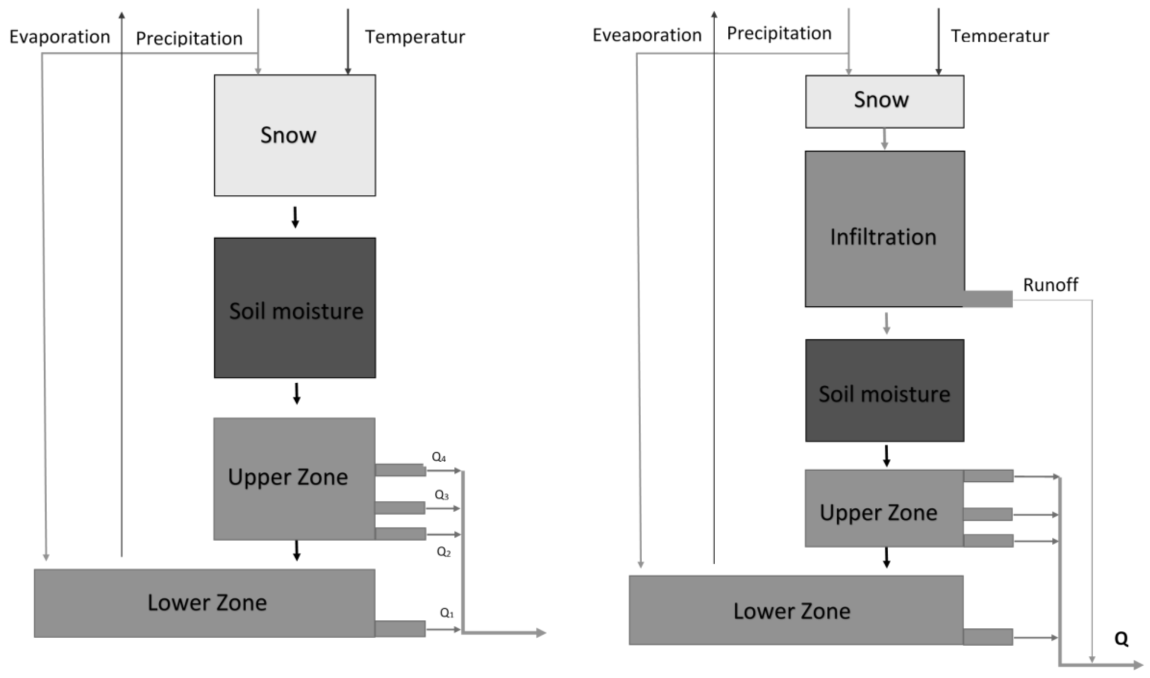
4.1.1 Applicability in urban environments

The original HBV model was developed for natural fields, e.g. large river systems with a sub-basin size of 40 km^2. The original version of the HBV model does not distinguish between surface water runoff and soil water runoff (Lindström et al., 1997). The setup in Figure 4.1 shows a modified version of the HBV-model which considers all the water that hits impervious surfaces to generate runoff. For pervious surfaces, it sets a threshold-value. Rainfall intensity exceeding this value will generate runoff (Hundecha and Bárdossy, 2004).

The latter approach is better to describe the hydrology in urbanized watersheds than the original HBV model. However, the mechanisms of infiltration make it hard to set a threshold value for infiltration. As described in section 0, the infiltration capacity, and therefore the magnitude of runoff generated from pervious surfaces, varies greatly depending on the soil moisture content level. Several hydrological models use an approach like the one in Figure 4.1. This thesis suggests an improvement to the HBV model by adding an infiltration routine between the snow routine and the soil moisture routine calculating the runoff, instead of setting a constant threshold value.

4.2 A new infiltration routine

The original HBV model considers runoff only form the soil routine as shown in Figure 4.2.a. Because of great variations in infiltration capacity in urban surfaces, consideration of the infiltration process becomes more important in urbanized watersheds than in natural dominated watersheds. An implementation of an infiltration routine is therefore suggested as shown in Figure 4.2.b.



a) **b)**

Figure 4.2 *The HBV-model with and without an infiltration routine*

Input precipitation will normally go through the snow routine, or if no snow, directly to soil moisture routine. The new infiltration routine will be placed between the snow routine and the soil moisture routine as shown in Figure 4.2.b. The infiltration routine will have two output values; infiltrated water, and runoff water. Infiltrated water will be forwarded as input to the soil moisture routine. Further calculations through the other routines will proceed as in the original model but with less water due to the loss of runoff water in the infiltration routine. Depending on the properties of the evaluated subbasin, runoff water could either pond on the surface and infiltrate when the rainfall intensity drops, or it can flow to the neighbouring downstream subbasin.

4.2.1 Input data in the infiltration routine

For the infiltration routine to be useful in a full hydrological model, it should be feasible to obtain the input data for the routine without investing too much time. Table 4.1 displays the input parameters necessary to implement the GARTO approach. All these parameters are additional parameters to the already existing-parameters in the HBV model.

Tabell 4.1 Input parameters in the infiltration routine

Physical description	Symbol	Unit
Bubbling pressure/air entry pressure	ψ_b	[L]
Pore distribution index	Λ	[-]
Saturated volumetric soil moisture content	θ_s	[-]
Residual volumetric soil moisture content	θ_r	[-]
Initial volumetric soil moisture content	θ_i	[-]
Saturated hydraulic conductivity	K_s	[LT^{-1}]

The bubbling pressure (ψ_b) and the pore distribution index can be found using a technique described by Brooks and Corey (1966) (Brooks and Corey, 1966). Saturated and residual volumetric soil moisture content (θ_s and θ_r) and saturated hydraulic conductivity (K_s) can be decided by field observations and lab tests. Initial volumetric soil moisture content (θ_i) can be guessed without further investigations. This is because initial soil moisture content will change during the simulations, and the start value I therefore not of significantly importance for a longer simulation periods.

Alternatively, these parameters can be estimated based on literature. If the soil types are known, tables from e.g. Lai et al. (2015) can provide values for these parameters. The uncertainty in the calculations will however increase by applying this technique compared to using filed observations.

4.3 Statkraft Hydrological Forecasting Tools (SHyFT)

SHyFT is an open source hydrological toolbox developed by Statkraft. In SHyFT, a HBV model is implemented. The HBV model contains the routines actual evapotranspiration, snow, soil, and tank (see Figure 4.2.a). Each routine is programmed separately in the programming language C++ and merged in a HBV-stack code. The interface between the routines is programmed in Python. A new infiltration routine should be implemented as an own routine and merged with the other routines between the snow routine and the soil routine in the HBV stack.

A routine in SHyFT divides variables into three types: *parameters, states,* and *responses*. The parameters contain variables that remain constant throughout the simulation, e.g. soil properties. The states contain variables that change during the simulations, e.g. soil moisture level and infiltration capacity. The responses contain variables that are forwarded to the next routines, e.g. runoff and infiltrated water. Each routine in the HBV model in SHyFT works as a "black-box", and calculates state, response, and parameters hidden for the user. A routine provides however output values. Time and input data such as precipitation, potential evapotranspiration and air temperature are handled outside the routines and fed into each routine when needed.

4.3.1 C++ programming language

The C++ programming language originated from the C programming language. C is a general-purpose language that can write any kind of program. C++ is similar to C but it is object orientated and uses *classes*. A program in C++ consists of a main function "main" and classes. "Main" is stored as "main.cpp". A class contains member functions and member variables. A class consists of one header file "<name>.h" and one cpp-file "<name>.cpp". In the header file, the class member variables and the functions are initialized and stored, while the calculations are done in the cpp-file. When the program is run, "main" is called. "main" may again call on classes connected to "main". A class can take input variables from "main", run through a member function, store values to member variables and then return member variables to "main" (Savitch, 2010).

Method

5 Code development

The author of this thesis did not have any prior knowledge of programming. To understand the infiltration routine and gain programming knowledge, an own C++ code were first made for the infiltration routine. When this code gave satisfactory results, it was implemented in SHyFT. The technique used to develop the program was to start by developing a basic program and add on new features as programming skills were gained. This way the code developed to handle increasingly advanced rainfall events.

5.1 The prototype

In the *specialisation project* mentioned in section 1, a simple prototype of the GAR method was developed in excel. It was able to consider short rainfall events (Nyhus, 2016). GAR was used to calculate infiltration followed by redistribution. A copy of this prototype was developed in C++ as a basis for further development. This code considered one short rainfall event followed by redistribution and used a while loop to iterate infiltration. The precipitation data was stored in a vector. The rainfall event considered in the prototype was short and artificial. The prototype could not run multiple rainfall events. Figure 5.1 displays a flow-chart of the prototype code.

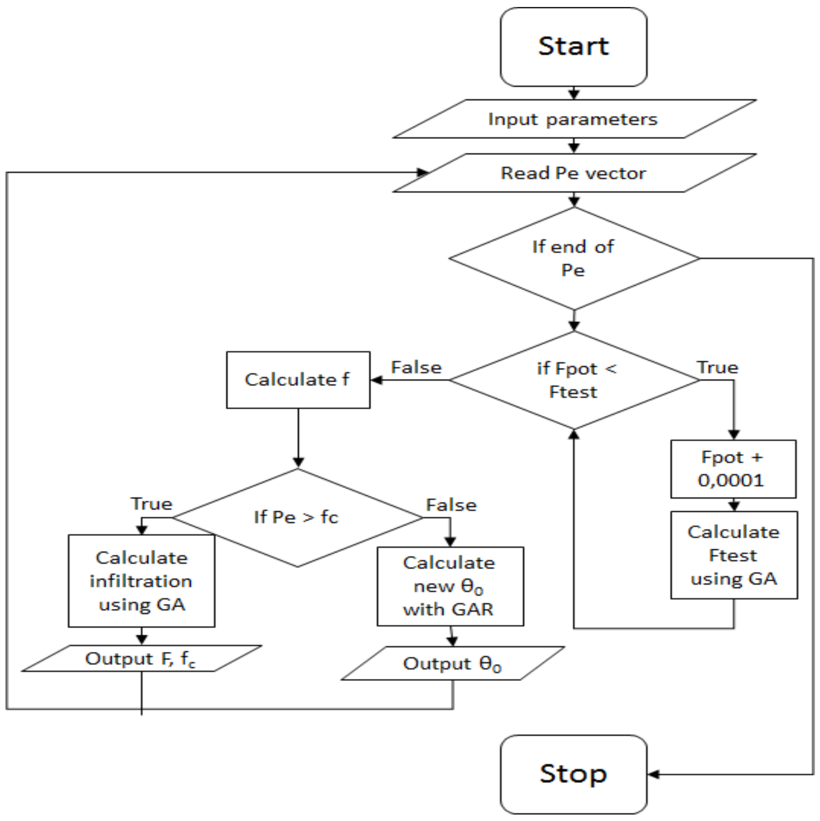


Figure 5.1 *Flow-chart of the prototype C++ code. From the specialisation project*

GA was used for calculations of *F* and f_c and the results were compared with results from Becker (2016) (Becker, 2016). GAR was used to calculate redistribution but there were no available results to validate the redistribution period for such a basic code.

5.2 Developing the final code from the prototype

It was not clear how GARTO dealt with several of the issues described later in this section. For further development of the prototype, a trial and error approach was therefore applied. To check whether the result became better it was compared to the results presented in GARTO.

Lai et al (1997) did four tests comparing the GARTO algorithm with the RE for eleven soil types.

Test 1 was chosen for comparison because this was the simplest of the four tests to simulate. Of the eleven soil types, four soil types had results for the wetting front depths, ponding time, deponding time and infiltrated water. Of these four soil types, soil type 4 and 11 were chosen as reference soils. The code was developed with soil 4 as standard soil and tested regularly for soil 11 to check that the code managed more than one soil type. When testing for several types of soils and rainfall intensities, a wider understanding of problems with the code was gained and several issues were solved.

Section 5.2.1-5.2.4 describes the steps of how some main issues were solved to go form the prototype to the finished code.

5.2.1 Mistaken interpretation of the capillary drive.

During coding process of the prototype in C++, it was discovered an error in the excel prototype. *Z* was multiplied with timestep (delta t) in Equation 7, which it should not have been. The error was probably a result of a misunderstanding of the mechanisms of capillary drive *G*. GAR includes Equation 10 where the bubbling pressure ψ_b is negative which makes *G* negative (Ogden and Saghafian, 1997). However, GARTO describes *G* in Equation 11 which implies that *G* should be positive (Talbot and Ogden, 2008):

$$G(\theta_i, \theta) = \frac{-\psi_b}{\lambda}\left(\frac{\Theta^{3+1/\lambda} - \Theta_i^{3+1/\lambda}}{3 + 1/\lambda}\right) \tag{10}$$

$$G(\theta_i, \theta) = |\psi_b|\left(\frac{\Theta^{3+1/\lambda} - \Theta_i^{3+1/\lambda}}{3\lambda + 1}\right) \tag{11}$$

Here Θ is relative volumetric soil moisture content and λ is the pore distribution index.

5.2.2 Classes

An early version of the code was written in “main” without any connecting classes. Variables were initialized, calculated, and rewritten in a for-loop in the same file. This approach had two main disadvantages: (1) The code became long and messy resulting in debugging becoming hard because it was hard to discover where a problem occurred in the code; (2) The code became unstable because all calculations had access to all the variables. Variables could therefore be changed several places in the code and it was hard to understand when and where this happened.

Classes were introduced to solve these problems. “main” contained the structure of the code while it called on classes to calculate side operations as explained in section 6.1. this lead to the following advantages: (1) If a problem was connected to the structure of the code, “main” could be examined, while a problem connected to a specific operation could be examined in the class handling this operation. Both “main” and the classes became shorter and easier to understand; (2) Using classes, variables could be stored as private. As “Private”, only functions inside a class have access to the variables in that class. “main” or other classes could not change these variables without calling on functions inside that class. This gave the programmer more control over the variables and made the code more stable.

5.2.3 Redistribution and advance of wetting fronts

In an advancing wetting front, Z changes with the value ΔZ over one timestep. This can be described by the following equations(Lai et al., 2015):

$$\frac{dZ_k}{dt} = \frac{K(\theta_k) - K(\theta_{k-1})}{\theta_k - \theta_{k-1}} \left(\frac{G(\theta_i,\ \theta_0)}{Z_k} + 1 \right) \tag{12}$$

$$(\theta_0 - \theta_i) \frac{dZ}{dt} = \left(\frac{K_s G(\theta_i,\ \theta_0)}{Z} + K(\theta_0) \right) \tag{13}$$

How ΔZ can be calculated in a situation where the precipitation rate (*Pe*) is less than f_c is not stated clearly in the GARTO article. However, for situations where *Pe* exceeds K_s, and the last wetting front is saturated, Equation 12 should be used. It is also stated that Equation 12 is a variation of Equation 13 which comes from GAR. Therefore, it was assumed that Equation 13 should be used to calculate *Z* for the latest wetting front in an unsaturated situation and were *Pe* is less than K_s.

The infiltrated water depth *F(t)* is the total mass of infiltrated water during a rainfall event. *F(t)* can be calculated independently of $\Delta\theta$ and Z as described in section 5.2.4. Equation 14 shows a mass relationship between *F(t),* the wetting front depth *Z,* and the soil moisture content θ. The right side of the equation describes the volume of a wetting front as illustrated in figure 3.4.b.

$$F(t) = (\theta - \theta_i) * Z(t) \tag{14}$$

When moving from one timestep to the next the infiltration depth Z and the soil moisture content θ changes. In the prototype, when the rainfall intensity was low, Z from the last timestep was used as input parameter in the redistribution equation (Equation 7) to calculate the change in soil moisture content $\Delta\theta$. The last θ plus $\Delta\theta$ was then used in Equation 13 to calculate a new Z. By this approach the water mass balance described in Equation 14 was not retained. Since the variables Z and θ are dependent on each other the solution was to do an iteration.

Two approaches to iterate *Z* and θ were tested: In approach (1) *Z* was increased by incremental timestep and inserted in Equation 7 to calculate a new θ. The right side of Equation 14 was calculated with new values of Z and θ. If the water mass was below *F(t),* the iteration process continued. Approach (2) used the same approach as in (1), but ere θ was incrementally increased and used in Equation 13. See Figure 6.4 and 6.5 for flow chart and code of this process.

To decide which approach that gave the best results, a trial and error approach was applied. It was found that approach (1) gave best results when $Pe < K_s$ and approach (2) gave best results when $Pe > K_s$.

5.2.4 Infiltration

The prototype used Green-Ampt (GA) to calculate infiltration depth *F*, and infiltration capacity f_c. A rainfall intensity greater than the infiltration capacity is a prerequisite for GA. A natural precipitation event will rarely start with such high intensities. To develop the code to handle all types of precipitation events, several changes where done.

ftest, was created as a test variable to decide ponding time. In each timestep in situation B, *F* increased with *Pe* and was used as input parameter in GA (Equation 4), to calculate *ftest*. In the beginning of the next timestep *Pe* was compared to *ftest*. If *Pe > ftest* situation B ended and situation C started.

In situation C, GA was used for further infiltration calculations without including cumulative infiltrated water *F* from situation B. This could be interpreted as shifting the curve D to the end of curve B in Figure 3.1. This approach gave a too high f_c resulting in a too high *F*.

Mein and Larson (1973) developed an approach to shift curve D to start at the end of situation B and at the same time account for the accumulated infiltration depth *F* during situation B. The approach is often referred to as Green-Ampt Mein Larson (GAML) (Chu, 1987). The method calculates time at ponding t_p and cumulative infiltration depth F_p at t_p. For a certain *F*, a time *t* can be decided based on these two variables. If *t* is higher than t_p, ponding occurs and the rainfall event change form situation B to situation C. GAML solved the problem and calculations of f_c became accurate for all situations A-C. The method follows the equations below:

$$t = tp + \frac{1}{K_s}\left[F - F_p + |\psi_f|(\theta_s - \theta_i)\ln\left(\frac{|\psi_f|(\theta_s - \theta_i) + F_p}{|\psi_f|(\theta_s - \theta_i) + F}\right)\right] \qquad (15)$$

$$F_p = \frac{|\psi_f| K_s(\theta_s - \theta_i)}{Pe - K_s} \qquad (16)$$

$$t_p = \frac{F_p}{Pe} \qquad (17)$$

Sand where tested in test 2 in GARTO described in 0. For the last wetting front the infiltration rate for the fifth pulse did not match with GARTO. It was discovered that the rainfall rate in the beginning of the pulse exceeded the infiltration capacity. GAML is not applicable for this situation, but GA is. GA was therefore used when time of ponding were at the beginning of a rainfall event.

6 Code structure

The developed C++ code of the infiltration routine consists of “main” and six classes. “main” initializes and keeps track of all states and responses. Further it controls whether the infiltration follows the situation A, B, or C in Figure 3.1. the code also imports precipitation data and stores it in a vector. The classes calculate and changes certain parameters, states, and responses, and return them to “main” as explained in section 6.2. the six classes are called “Infiltration”, “getPrecipitation”, “infiltration_depth”, “Parameters”, “Ponding” and “soilMoitsture”.

Table 6.1 and 6.2 explains the parameters, states, responses, outputs, and other variables used in the code. When developing the code, it was not necessary to distinguish on states, parameters, or response. This became important when implementing in SHyFT. However, it gives a clearer picture on the nature of a variable used in the code.

Tabell 6.1 *Parameters, States, and responses*

Physical description	Symbols in thesis	Symbol in the code	Unit	Classification
Green Ampt wetting front soil suction	*Hc*	*Hc*	[L]	Parameter
Bubbling pressure/air entry pressure	ψ_b	*Yb*	[L]	Parameter
Pore distribution index	*Λ*	*lambda*	[-]	Parameter
Saturated volumetric soil moisture content	θ_s	*Os*	[-]	Parameter
Residual volumetric soil moisture content	θ_r	*Or*	[-]	Parameter
Initial volumetric soil moisture content	θ_i	*Oi*	[-]	Parameter
Saturated hydraulic conductivity	K_s	*Ks*	[LT^{-1}]	Parameter
Initial hydraulic conductivity	K_i	*Ki*	[LT^{-1}]	Parameter
Relative volumetric initial water content	Θ_i	*ORi*	[-]	Parameter
Relative volumetric water content	Θ	*OR0*	[-]	Parameter
Number of wetting fronts	*K*	*k*	[#]	Sate
Current hydraulic conductivity	$K(\theta)$	*K0*	[LT^{-1}]	State
Current volumetric soil moisture content	θ	*O0*	[-]	state/output
Infiltrated physical depth	*Z*	*Z*	[L]	state/output
Actual infiltration	*f*	*f*	[LT^{-1}]	state/output
Ponding		*ponding*	[L]	state/output
Infiltrated water depth	*F*	*Freal*	[L]	response/output
Runoff		*Runoff*	[LT^{-1}]	response/output

Table 6.2 *Other variables in the code*

Description	Code symbold	Unit
Water demand for wetting front 0-(k-1)	*Pek*	[L]
Residual rain water	*PeR*	[L]
Porewater volume for wetting front 0-(k-1)	*Vzk*	[L]
Unsaturated capillary drive	*G*	[L]
Infiltration rate if $Pe > f$ since the start	*ftest*	[LT^{-1}]
Infiltrated water before the start	*deltaF*	[L]
Rainfall rate in last timestep	*PeL*	[L]
Soil water content during iteration	*O0test*	[-]
Wetting front depth during iteration	*Ztest*	[L]
Infiltrated water during iteration	*Ftest*	[L]

6.1 Main

Main initializes variables and decides which of the situations A, B or C that is present. "Main" is structured into two parts: (1) Initialization of parameters states and responses and (2) A for-loop running through the precipitation series, consisting of three parts.

In part (1), the classes are called and the class objects are given names. A precipitation vector called *Pe* is initialized to a zero vector with a length equal to the length gathered from the precipitation class. Each value of the precipitation series is stored in the vector *Pe.* Parameters are initialized to the values stored in the class "Parameters". Time, cumulative infiltration *Freal*, infiltration rate *f*, the test parameter *ftest*, number of wetting fronts *k*, ponding and precipitation intensity from last timestep *PeL* are set to zero. The six vectors for soil moisture content *O0*, relative volumetric soil moisture content *OR0,* conductivity *K0,* wetting front depth *Z,* water volume in wetting front k *Vzk* and water demand of wetting front k *Pek* are created to store the states of each wetting front. Each vector is assigned four elements where the first value is initialized to initial conditions of wetting front zero, and the other values are set to zero.

(2) The for-loop changes the parameters, states and responses and controls which of the situations A, B, and C that are present. The loop is divided into three parts. The first part (i) controls the advance of wetting front 0-(k-1), prat (ii) decides which situation A-C that is present and part (iii) calculates K0 and OR0 and merges wetting fronts if necessary.

Part (i) describes the advance of the wetting fronts that are not under development, wetting front 0 - (*k-1*). The flow chart in Figure 6.1 describes the process. For these wetting fronts θ is constant and *Z* is increasing. The wetting fronts therefore needs a supply of water to maintain mass balance. This water is taken from either rainfall water, ponded water, or the developing wetting front respectively. For each new timestep, the vectors *Pek* and *Vzk* are initialized to zero vectors. Residual rainwater (*PeR)* is initialized to *Pe* for the given timestep. A fore-loop runs through wetting front 0 - (*k-1*). For wetting front *j*, the pore water volume before an advance *Vzk* is calculated by multiplying Z and θ. The class "infiltration_detpth" is called and returns Z_{new} which is the depth for the next advance. The water demand for each wetting front (*Pek)* is calculated as the new volume, Z_{new} times θ, minus *Vzk*. *PeR* is calculated as *PeR-Pek* in each loop. This approach is repeated for all the wetting fronts 0 - (k-1). The remaining water for the developing wetting front *PeR* = ($\sum_{j=0}^{k-1} Pek[j]$) is used when calculating the advance of wetting front *k* in situation A.

(ii) Four if statements decide which of the situations A, B, or C are present or if a new wetting front is formed (see lower left part flow chart Figure 6.1). Each loop runs through all the if statements but only one of them will be true for each timestep. A new θ and *Z* are calculated differently depending on which of the if statements that is true as in section 6.1.2 to 6.1.4.

Part (iii) is the end of the for loop. It calculates a new *OR0* and *K0* and checks if one wetting front has exceeded the depth of another (see left part of flow chart in Figure 6.1). If so these wetting fronts are merged. Equation 12 is used to calculate a new *Z*. Further $\theta_{k-1} = \theta_k$ and *k* = *k-1*.

At the end of the loop, the code prints some information to the screen and all the information to a text document. The data is further imported to excel to do analysis of data.

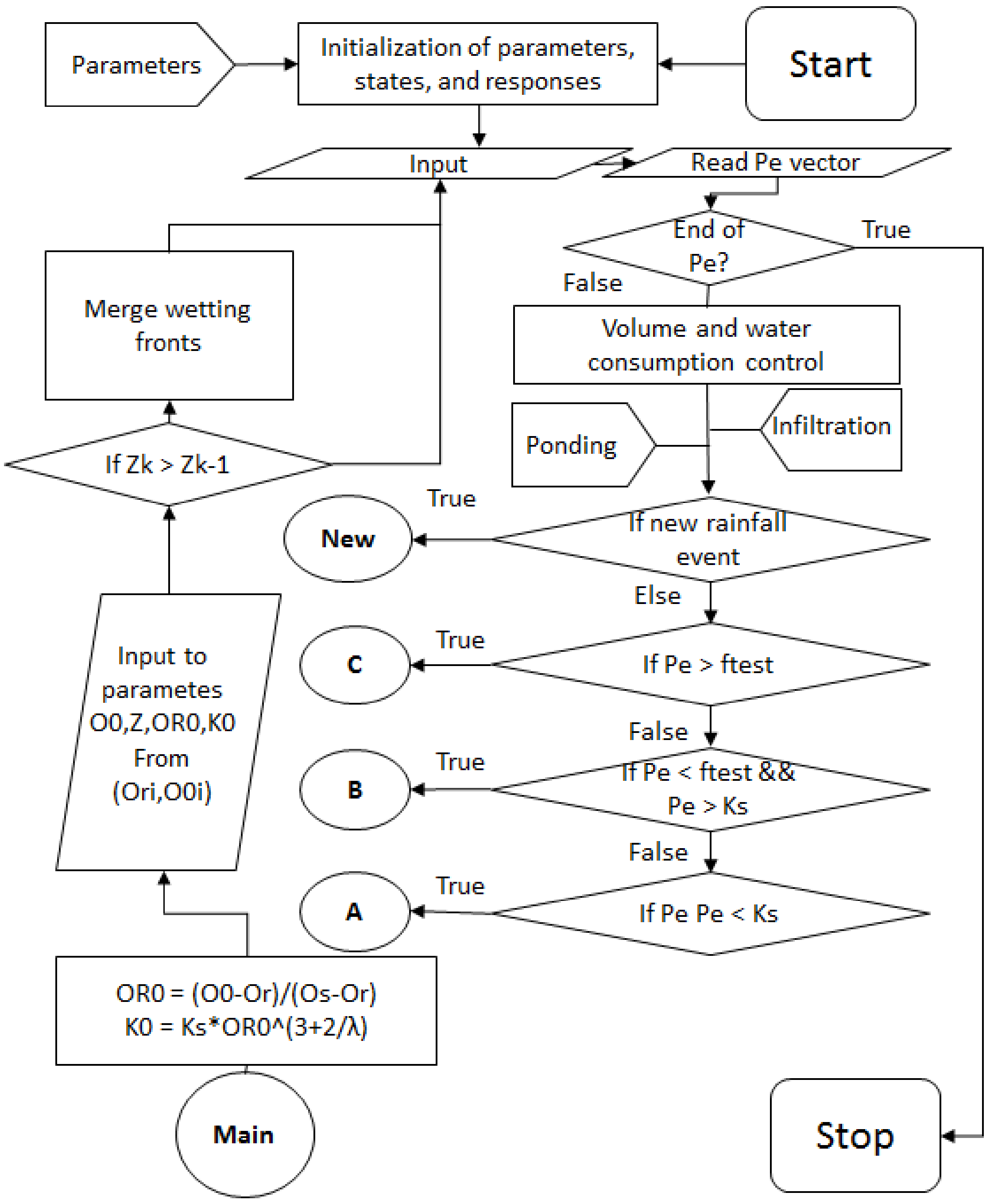
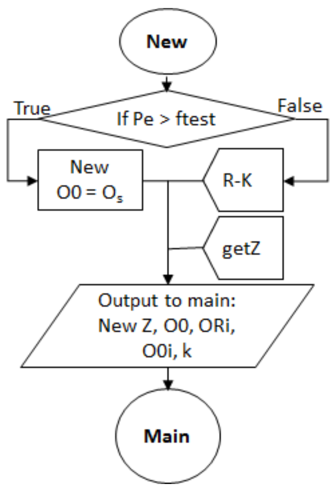


Figure 6.1 *Flow-chart of "main"*

6.1.1 New wetting front

If the precipitation is higher than saturated conductivity, and the precipitation in last timestep is lower than saturated conductivity, a new wetting front is formed (Figure 6.2). The if statement also checks whether θ is less than θ_S. This check is only important if ponding is allowed. If ponded water form last rainfall event still is infiltrating, no redistribution has yet occurred, and a new wetting front will not be created. If the if statement is true, the class “Infiltration_depth” calculates an initial depth. Another if statement checks if the *Pe* exceeds *ftest*. If so θ is set to θ_s. If not, the class “SoilMoisture” calculates a new θ value.

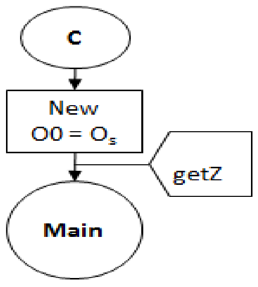


```
if (Pe[i] > Ks && PeL < Ks && O0[k] < Os){
    getZ.setZK0(O0[k-1],OR0[k-1]);
    Z[k] = getZ.getZK0();
    if (Pe[i] > ftest){
        O0[k] = Os;
    } else{
        output.setO0(Z[k],PeR,O0[k-1],OR0[k-
1]);
        O0[k] = output.getO0();
    }
}
```

Figur 6.2 *Flow-chart describing the creation of a new wetting front*

6.1.2 Situation C

Situation C, Figure 6.3, is the present situation if the precipitation rate *Pe* > the maximum infiltration capacity of the soil *ftest.* Ponding is present occurs in situation C and the soil reaches is in a fully saturated state, $\theta = \theta_s$. A new Z is calculated using a saturated form of Equation 13 which is calculated in the class “Infiltration_depth”. θ and Z are returned to “main”.

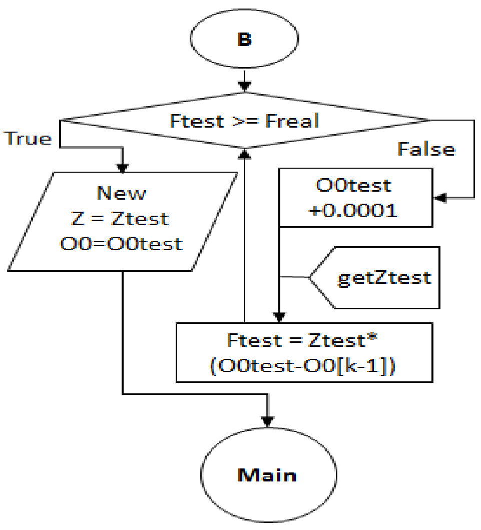


```
if (Pe[i] >= ftest){
      getZ.setZs(O0[k-1],Z[k]);
      Z[k] = getZ.getZs();
      O0[k] = Os;
}
```

Figure 6.3 *Flow-chart describing situation C*

6.1.3 Situation B

If *Pe* is less than f_c, but higher than K_s, Situation B is present. The infiltration is then calculated as shown in Figure 6.4 Three test parameters, *Ftest*, *Ztest* and *O0test*, are created to do the iteration explained in 0. They are initialized to the values from last timestep. The amount of infiltrated water *Freal*, will in situation B equal total rainfall during the rainfall event. The parameter *deltaF* is gathered form the infiltration routine. *deltaF* offsets *Freal* to only measure infiltrated water during the last rainfall event in case of multiple precipitation events. A while loop with the decision criteria "is *Ftest* >= Freal" is used to iterate new values for θ and *Z*. O0test is increased with incremental steps. The class "Infiltration_depth" is called to calculate a new *Z*. *Ftest* is calculated as the new volume of the wetting front *Z* times $(O0 - O0)$. Ftest is compared with *Freal*. If *Ftest* < *Freal* the while-loop continues. *Ftest* >= *Freal*, the while loop ends. A new Z is set equal to *Ztest*, and *O0* is set to *O0test*. These values are then returned to "main".



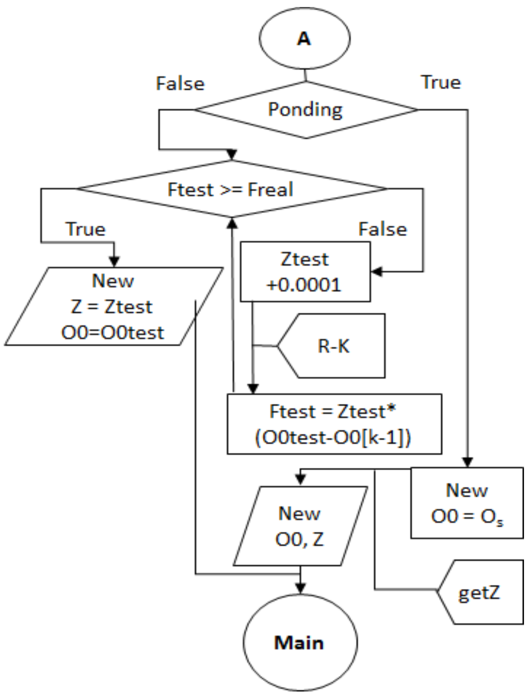
```
float O0test = O0[k];
float Ztest = Z[k];
if (Pe[i] < ftest && Pe[i] >= Ks) {
   float Ftest = Freal - deltaF –
   f*timestep;
   while (Ftest < Freal - deltaF) {
     O0test = O0test + 0.0001;
     getZ.setZK1(K0[k],O0test,O0[k-1],Z[k],
     OR0[k],OR0[k-1]);
     Ztest = getZ.getZK1();
     Ftest = Ztest*(O0test – O0[k-1]);
   }
   Z[k] = Ztest;
   O0[k] = O0test;
}
```

Figure 6.4 *Flow-chart describing situation B*

6.1.4 Situation A

If *Pe* is less than K_S, situation A is present, and Figure 6.5 describes the calculation process. However, an if statement considers whether ponding still occurs. If ponding occurs, the code repeats the calculations in situation C.

If there is no ponding, iteration like the one in situation B is stared. However, the iteration object is now *Z* and not θ as explained in section 0. The variable *Ztest* is increased with incremental values and *O0test* is gathered from the class "soilMoisture". *Ftest* is calculated as in B and the criteria *Ftest>=Freal* is used to continue or end the while loop, like in B. New values of *Z* and *O0* are set equal to *O0test* and *Ztest*, and returned to "main".



```
if (Pe[i] < Ks){
    if (ponding > 0){
        getZ.setZs(O0[k-1],Z[k]);
        Z[k] = getZ.getZs();
        O0[k] = Os;
    else {
        float Ftest=Freal-deltaF-0.001;
        while (Ftest < Freal) {
            Ztest = Ztest + 0.001;
            output.setO0(Ztest, PeR,
            O0[k], OR0[k-1]);
            O0test = output.getO0();
            Ftest=Ztest*(O0test-O0[k-1]);
        }
        Z[k] = Ztest;
        O0[k] = O0test;
    }
}
```

Figure 6.5 *Flow-chart describing situation A*

6.2 Classes

As explained in 5.2, the classes contain functions to calculates some variables, and they also store variables as private to prevent “main” to change the variables without permission. Two functions are normally needed to change and return a variable from a class. A “set-function” is called by “main” and will calculate a new value for a variable and store it as private. Then “main” call a “get-function” to return the new value of the variable to “main”.

6.2.1 The class “getPrecipitation”

The class “getPrecipitation” reads and returns the length of the precipitation file. “getPrecipitation” contains the functions “setLength()” and “getLength()”. “setLength()” reads the precipitation file named in “Parameters”, which is stored in a text document. Further it counts the number of values stored in the file. This number is stored in a private variable *length*. “getLength()” returns *length* to “main”. “main” uses this number to initialize a zero vector, *Pe*, with the length “length”. A while-loop in “main” runs through the text document again and stores each precipitation data in the right order in the vector *Pe*.

6.2.2 The class "Parameters"

The class "Parameters" contains parameters and initial states for the soil of consideration. All the variables in the class "Parameters" will stay the same throughout the simulations. The class "Parameters" does therefor not have any "set-functions". This means that "main" cannot change any variables given in the class "Parameters". "main" is only allowed to return parameters from the class "Parameters". This makes the code more robust.

The user initializes the states and parameters marked with "user input" in the table below. The class calculates the states and parameters marked with "calculated".

Tabell 6.3 *Parameters and state stored in the class "Parameters"*

Description	Code symbols	Unit	Classification
Bubbling pressure/air entry pressure	ψ_b	[L]	Parameter (user input)
Pore distribution index	Λ	[-]	Parameter (user input)
Saturated volumetric soil moisture content	θ_s	[-]	Parameter (user input)
Residual volumetric soil moisture content	θ_r	[-]	Parameter (user input)
Initial volumetric soil moisture content	θ_i	[-]	State (user input)
Is ponding Allowed	*Pond*	[char]	State (user input)
Saturated hydraulic conductivity	K_s	[LT^{-1}]	Parameter (user input)
Green-Ampt wetting front soil suction parameter	*Hc*	[L]	Parameter (Calculated)
Initial hydraulic conductivity	K_i	[LT^{-1}]	State (Calculated)
Relative initial volumetric water content	Θ_i	[-]	State (Calculated)

6.2.3 The class "Infiltration"

The class "Infiltration" is a crucial class for the code. It calculates and returns *F, deltaF, ftest,* and f_c. These parameters are essential to control the if statements in "main" and to monitor the responses from the observed area. *F* is the infiltrated water depth since the first rainfall event started. *f* is the current infiltration rate. *ftest* is the potential infiltration capacity. *deltaF* is the total infiltrated water before the current rainfall period started. The class "Infiltration" can also return the parameter *error* which is explained later.

"Infiltration" is called by the function "setFf()" which takes four input arguments, *time, Pe, ponding, k, O0*(k-1), and *i*. "setFf()" calculates *F, deltaF, ftest,*and f_c and stores them in private variables. To return these values the functions "getf()", "getftest()", "getF()", and "getdletaF()" are called. Since *deltaF* is stored in a vector "getdletaF()" takes the input argument k to determine which of the values in the vector that are returned.

The infiltration routine consists of two parts. (1) The test parameters *Ftest1* and *ftest* are set. *Ftest1* is *F* plus *Pe*. *ftest* is calculated with *Ftest1* and is the maximum infiltration capcacity for the soil. This parameter is used by “main” to decide when ponding occurs. When a new wetting front is formed, several parameters is reset. “main” needs to keep track of total infiltration depth *F* across the whole time-series, but to be able to calculate infiltrated water during the last rainfall event with GAML, *F* and *time* needs to be set to zero. *deltaF, deltatime* and *ktest*, where introduced as member variables in the class “Infiltration”. In “main” *k* controls how many wetting fronts that are present. Therefore, *k* where exposed to “infiltration”. In “Infiltration” an if statement controls if *k* increases or decreases over the last timestep and updates *ktest* to the new *k*. If *k* increases *deltaF* and *deltatime* is set to current *time* and *F*. *O0[k-1]* is also stored as *O0L*. Calculation of infiltration subtract *deltatime* and *deltaF* so that every new wetting front starts infiltration calculations at the time when the rainfall event started and with no infiltrated water.

(2) In part two, infiltration depth at time of ponding (*Fp)* and time of ponding (*tp*) is calculated as in Equation 16 and 17. A test parameter for iteration (*Ftest*) is initialized. Four if statements decide how *F* and *f* should be calculated. (i) If there is raining, but the intensity is low, all the water is infiltrated and *f* is set to the rainfall intensity. (ii) If the rainfall intensity exceeds *ftest* at the first timestep, GA can be used directly to calculate *f* and *F,* without applying Mein and Larson’s approach. (iii) If *Pe* is greater than *Ks*, but less than *ftest,* GAML is used. GAML iterates a parameter (*t*) which is the time calculated based on infiltrated water. Further an if statement decide if *t* is less than *tp*. If so all the water infiltrates. If not, *F* and *f* is calculated by using GA. (iv) The fourth if statement I true if water is ponded on the surface. Then if statement three is continued even if it is not raining until all the ponded water is infiltrated. If one rainfall event follows close after another so there is not enough time for the water to infiltrate, the soil would experience that there is one long rainfall event because of the fourth if statement. Only one wetting front is formed. If none of the if statements is true there is no rainfall and the *f* is set to zero. *F* remains unchanged. A flow-chart of the class “Infiltration” is shown below.

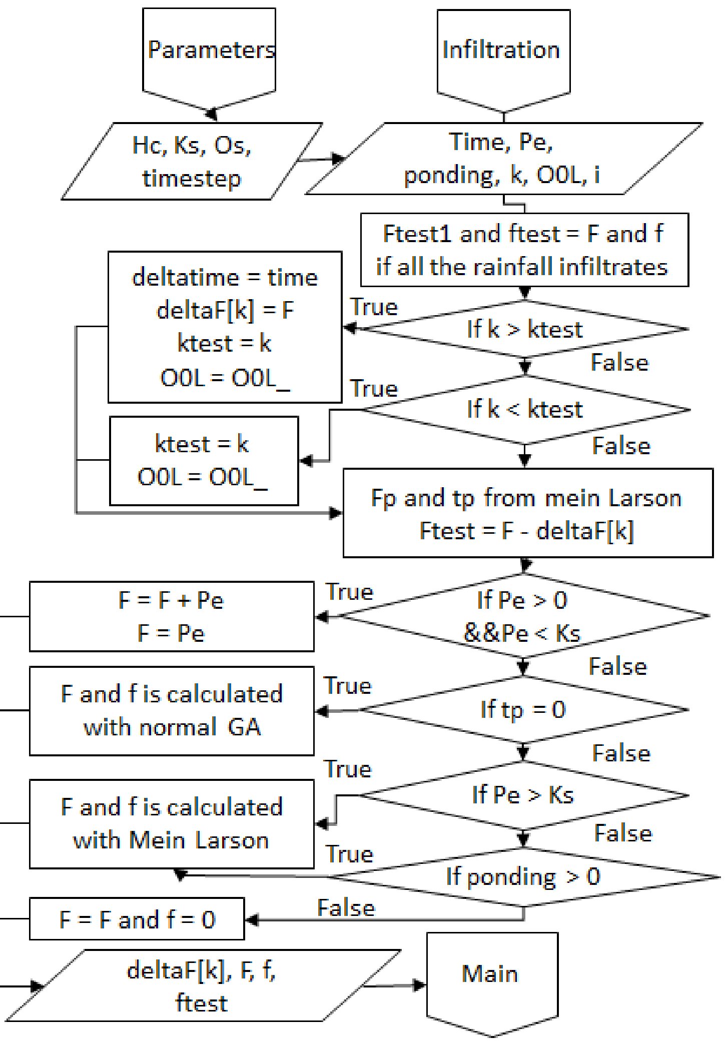


Figure 6.6 *Flow-chart of the class "Infiltration"*

When the depth of wetting front *k* exceeds the depth of wetting front *k-1* and the wetting fronts are merged the infiltration rate raised above the rainfall rate. The infiltration rate should in theory be the same over this transition from two wetting fronts to one, but because *O0(k-1)* changes over this transition and everything else is the same the infiltration rate increases. In the code the solution is to keep *O0(k-1)* from two wetting fronts.

6.2.4 The class "Ponding"

The class "Ponding" controls whether ponding occurs and how much water is ponded on for each timestep. It returns either *ponding* or *Runoff*. The class takes infiltration rate *f*, precipitation rate *Pe*, *timestep* and data number *i* as input variables. In "Parameters", the user has decided whether ponding is allowed. This information is read by "Ponding" and is fed into an if statement. If ponding is not allowed, runoff is calculated and ponding is set to zero and returns this information to "main". If ponding is allowed, two if-sentences check whether *Pe* is higher or lower than f_c, and calculates ponding as shown in Figure 6.7 If Ponding is calculated to less than zero, all the ponded water is infiltrated during the last timestep, and ponding is set to zero. The runoff is always zero if ponding is allowed. Ponding and runoff are returned to main.

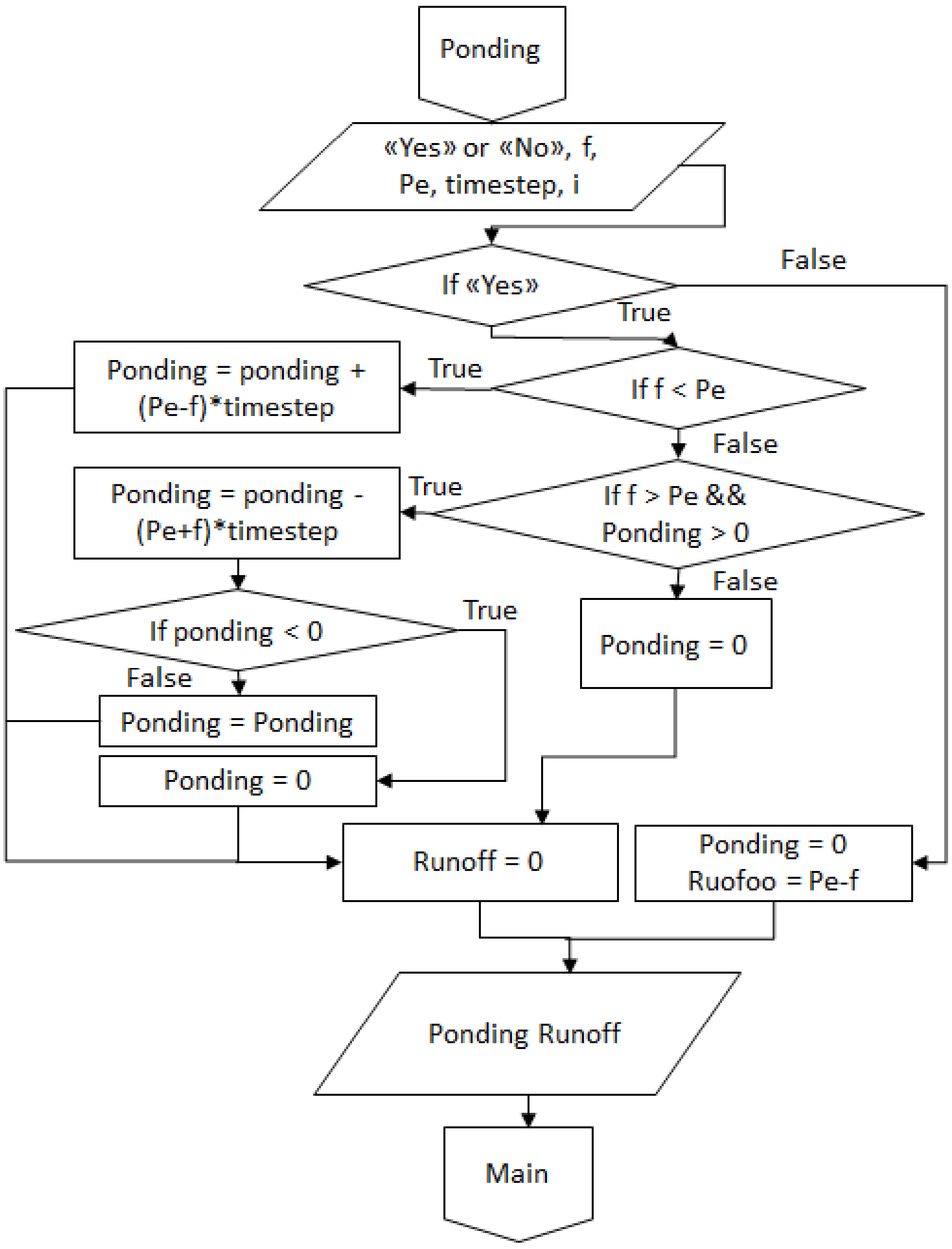


Figure 6.7 *Flow-chart describing the class "Ponding"*

6.2.5 The class "Infiltration_depth"

The class "infiltration_depth" calculates the advance of a wetting front in Z direction. The Runge-Kutta method (RK4) is used for numerical integration of ΔZ between each timestep. A new depth Z is set equal to the old value of Z plus ΔZ. By applying RK4, stability in the code is maintained and the code generates accurate results. The timesteps should not exceed 0.5

minutes when using RK4 (Lai et al., 2015) “Infiltration_depth” contains four separate ways of calculating the advance of a wetting front; (1) Equation 8 calculates “dry depth” which is the advance of a wetting front during the first timestep after it is formed; (2) Equation 12 is used for a situation where *Pe* is greater than f_c.; (3) When the soil is fully saturated, a saturated form of Equation 12 is used; (4) Equation 13 is used if there is only the initial wetting front and one additional wetting front.

Main needs to call the function “setZ” to set a new Z-value which is stored in private variables in the class “Infiltration_depth”. Another function “getZ” is called to return the stored Z-value. This is a safe way of changing and returning the stored Z-value without giving main the rights to modify private variables in the class “Infiltration_depth”.

6.2.6 The class “soilMoisture”

The class “soilMoisture” calculates the redistribution of soil moisture when the rainfall intensity is less than K_s. The soil moisture routine also applies RK4. *PeR* is used as input value for the precipitation. Then the coefficient p = 1 if *PeR* > 0, or 1,7 if *PeR* = 0. *O0* is calculated using Equation 7.

As in the class “Infiltration_depth”, the *O0* is set by the function “setO0” and it is returned by the function “getO0”.

6.3 Error messages

If input data is wrong, the program will produce results without value for the user. To avoid wasting time on producing these results and to give the user an indication of what the problem is, error messages are implemented. For certain cases where it is obvious that the user has entered wrong input data, the program will detect these mistakes and print a short message about what the problem might be. The error messages are printed form the classes “getPrecipitation”, “Ponding” and “Parameters”. The error messages are shown in table 6.4.

Table 6.4 *Error messages built into the program*

1	"ERROR: The typed file does not exist in the right folder"
2	"ERROR: Precipitation data at time " << (x+1)*0.008333 << "h is < 0"
3	"ERROR: Precipitation data at time " << (x+1)*0.008333 << "h has more than 8 decimals"
4	"ERROR: A Parameter in the class Parameters is < 0"
5	"EROOR: The variable Or > Oi or Oi > Os or Or < Os"
6	"ERROR: Or or Oi or Os >= 1"
7	"ERROR: Wrong spelling of Yes or No in class ponding"

Error message 1 in table 6.4 is returned if the user specifies an invalid precipitation file or the file is in the wrong folder and the program fails to open the file.

When the program stores precipitation data in the *Pe* vector, it will check whether any of the data has a value below zero. If this is the case, the program will exit and return error message 2 in table 6.4. The program cannot handle precipitation data with more than 8 decimals. If any of the data entered has more than 8 decimals the program will exit and return error massage 3 in table 6.4. Both these error messages will display the location in the data series where the mistake is detected.

In the class "Parameters", none of the parameters can have a value lower than zero. The soil moisture content parameters θ_s, θ_i, and θ_r must have a value between zero and one because they are a measure relative to total volume of the soil. The parameter θ_r cannot be greater than either θ_i or θ_s, and θ_i cannot be greater than θ_s. If the input values fail to fulfil any of these requirements, the program will exit and return the error message 4 to 6 in table 6.4.

The user needs to specify whether water will pond on the surface or create surface runoff. This is specified by typing "yes" or "no" in the class "Parameters". The program understands all combinations of capital letters and lowercase letters. However, if the user types anything else than no or yes, the program will exit and return the error message 7 in table 6.4.

7 Implementation of the code in SHyFT

When the code was developed and produced reasonable results compared to GARTO, it was implemented in SHyFT.

7.1 Compiling SHyFT and including the infiltration routine

SHyFT was installed and compiled on the computer by following the installation manual provided on the open source web based Git repository (Helset, 2017). An own branch in SHyFT, called “ntnu_hbv_inf”, was established to host the project of the infiltration routine.

A summary of the changes that were conducted is described in section 7.1.1. An overview of all the changed files and in which lines the changes are done, are attached in appendix C. In section 0, a description of the infiltration routine used to compile the code is described.

7.1.1 Changed files

In the file “core_serializtion” the infiltration routine was included by including the file “hbv_infiltration”. The states of the infiltration routine were introduced to the archive to store the states of the infiltration routine.

In the file “hbv_stack”, the infiltration routine was included by including the file “hbv_infiltration”. The parameter class of the infiltration routine was introduced as an object to expose the parameters to the file by the function “set”. The function “get” returns the parameters when called, and the function “get_name” returns the name of the routine. The states and responses were also exposed to the file. Thereafter, the states, parameters, responses, and the function “step” from the infiltration struct “calculator” in the file “hbv_infiltration” were exposed to the function “run_hbv_stack”. The variable “outflow”, in the snow routine, was redirected as input to the infiltration routine, while the input of the soil routine was replaced with the variable “Freal” from the infiltration routine. This way the HBV stack could stack the infiltration routine between the snow routine and the soil routine.

In the file “hbv_stack_cell_model”, the responses “Freal” and “Runoff” were included in the struct “all_response_collector”. This way, the time and location of the responses could be initialized and the responses could be collected in the function called “collect”. In the struct “state_collector” all states are collected. The states of the infiltration routine was included.

In the file “hbv_stack_test”, the namespace “infil” was set equal to “shyft::core::hbv_infiltration” in order to include the infiltration routine. The parameters were imported and the states were initialized.

7.1.2 Basic infiltration routine

To test whether the infiltration routine was properly implemented in SHyFT, a simple code was developed in the struct “calculator”. The files “hbv_soil” and “hbv_soil_test” were copied to separate files and they were modified to host the infiltration routine. A first edition of the infiltration routine was implemented and coded. The first edition contained only the parameter *Os*, the state *O0,* and the responses *Freal* and *Runoff.* In the file “hbv_infiltration” the struct “calculator” containing the function “step”, updated states and responses for each timestep. In this function, a simple code was developed as shown below.

```
void step(S& s, R& r, shyft::core::utctime t0, shyft::core::utctime t1, double
snow_out) {
        r.Freal = param.Os*snow_out;
        r.Runoff = 1;
        s.O0 = 1;
}
```

The aim of this simple code was to verify that the hbv-stack, including the infiltration routine, was running without errors. In the file “hbv_infiltration_test” tests were implemented to ensure that the user does not run a simulation with invalid input data. Tests that would be true based on the calculations in the function “step” were made. SHyFT was built and run, and compiling errors where rectified.

7.2 Implementation of the code

When the hbv-stack was compiling without the code returning errors, the code described in section 6 was implemented. The struct “calculator” was implemented in “hbv_infiltration_test” and the namespace was set to “hbv_infiltration2”. Under TEST_CASE(“test_regression”) the line including the calculator was referring to the object “hbv_infiltration2” and not “hbv_infiltration”. This way, only the “hbv_infiltration_test” needed to be compiled and run during debugging, and the process went therefore faster.

7.2.1 Modification of the code to match the SHyFT framework

The structure of the code implemented in SHyFT is similar to the structure described in section 6. However, the interpretation of time and the data handling mechanisms had to be changed to make the code fit the framework of SHyFT.

The interpretation of time is different in SHyFT and in the developed code. Precipitation data in the developed code is presented as a time independent data series. The code interpreter the time as a number and allocates the first data in the series the time number zero. For each new data, this number increases by a number given in hours, e.g. the second value in the precipitation data will get the number 0.00833 hours, which corresponds to half a minute.

In SHyFT, time is also interpreted as a number. However, the number is date dependent. SHyFT uses the computer's *system clock* that counts ticks from a decided starting time. Each tick is counted and the time number is given based on number of ticks since the starting time. A tick is equal to one second. When presenting a precipitation data in SHyFT, the date needs to be specified, so that SHyFT can decide the time number (how many ticks since starting time) it should allocate to this data.

Because of the different interpretation of time in SHyFT and the developed code, the for loop used to control time and precipitation data described in section 6, was not applicable in SHyFT. Because the time and input water is handled outside the routine, the routine in SHyFT is like a function that takes input water data, calculates new states and responses, and returns them to the global HBV model. The response "snow_out" from the snow routine was used as the variable "precipitation" was used in the code.

In the developed code, some calculations are separate classes and not included in "main". In SHyFT this is not an option. Some of the classes were therefore implemented directly into the function "step". Other classes which were called several times during "main", were implemented as own functions before the function "step" and were called whenever needed. The code in SHyFT is further explained in section 7.3.

In SHyFT, all variables are named with lower case letters and underscore between words. All the variables from the code where renamed to fit these rules.

7.3 The structure of the code in SHyFT

The struct "calculator" calls on the struct "parameter", "states" and "responses" to get the current values and to be able to change the states and responses.

The two classes "Infiltration_depth" and "soilMoisture" from section 6.2 are implemented as functions. Four functions describe the four different ways of calculating the infiltration depth

Z, as described in section 6.2.5, while one function calculates the soil moisture content θ as described in section 6.2.6.

The function “step” conducts all the other calculations and “step” is mainly a copy of “main” described in 6.1. However, the infiltration routine and the ponding routine are merged together and placed before the if statements. The if statements which decide the infiltration situation (see section 6.1) equals the if statements in the developed code.

The classes “getPrecipitation” and “Parameters” are not needed in SHyFT. The precipitation is controlled by SHyFT and is introduced to the infiltration routine as the response variable from the snow routine called “snow_out”. The parameters are replaced by the struct “parameters”.

7.4 Further work with SHyFT

In the file “hbv_stack_cell_model”, the responses from the infiltration routine might have been implemented incorrectly. The result of this would be that the HBV model will misinterpret the runoff. If this is the case, the runoff will probably not contribute to the total runoff from the evaluated catchment.

Because of problems with exposing the states stored as vectors, the struct calculator is still located in the file “hbv_infiltration_test”. The states need to be exposed properly so that all the files in SHyFT using states can read them. Then the struct “calculator” can be moved to the file “hbv_infiltration” where it should be located.

To initialize parameters as equations that calculates values using other parameters was difficult to implement. The parameters *ori, ki* and *hc* were therefore initialized to the fixed value of the equation using the soil type loam. To run the simulations for other soil types, these parameters must be properly implemented.

When the SHyFT has been compiled correctly with the infiltration routine properly included, the full HBV model should be tested for the same precipitation series as in Lai et al. (2015) to verify that the results from the infiltration routine match the results from the code.

The changes in infiltration depth Z and soil moisture content θ, is currently done by coded Runge-Kutta. There is available better and more stable Runge-Kutta functions. These would reduce the number of lines in the code and should be implemented.

Results and discussion

8 The code's performance

To investigate the code's performance, it was compared to results of GARTO method presented by Lai et al. (2015). Lai et al. (2015) compared results from GARTO to the results from FUCG RE. As explained in section 3.2.1, the RE method is considered to provide accurate results for infiltration simulations. Only comparing the results from the developed code against the GARTO model, and not against FUGG RE could cause cumulative error Nevertheless, the aim of the developed code is to reproduce the GARO model and the Lai et al. (2015) present more results for the GARTO method than for RE. To limit the study scope while at the same time comparing to as many results as possible, the results from the code are compared only to the results of GARTO.

The code where tested against the soil types loam, clay, and sand. Comparison of sand is only briefly presented. Soil properties for the three soils are listed in table 8.1.

Table 8.1 *Soil properties.*

Soil type	θ_r	θ_s	θ_i	ψ_b	λ	K_s
Loam	0.027	0.434	0.117	11.15	0.252	1.32
Clay	0.090	0.385	0.272	37.30	0.165	0.06
Sand	0.020	0.417	0.033	7.26	0.694	23.56
Green Roof	0.120	0.400	0.2302	8.00	0.600	43.928

Clay does have a low saturated conductivity, which means that rain water will pond with lower intensities. Sand has a high saturated conductivity which means that it ponds only on high rainfall intensities. Loam is between these two.

The developed code was compared to results from two scenarios in GARTO, test 1 and test 2, and results from a green roof in Trondheim. In test 1, a comparison of done both infiltration rate and infiltration depth where done.

8.1 Test 1

Test 1 describes rainfall sequences with two rainfall pulses, both with steady rainfall. The same rainfall event as used by Lai et al. (2015) was applied on the developed code. See Table 8.2 for duration and rainfall intensities for the rainfall events.

Tabell 8.2 Duration and rainfall intensity, Test 1

Soil	Pulse	Rainfall rate (cm/h)	Starting time (h)	Duration (h)
Loam	1	4	0	1
	2	4	3	1
Clay	1	1	0	1
	2	1	3	1

8.1.1 Infiltration rate

The developed code gives the same infiltration pattern as the results from Lai et al. (2015) (see figure 8.1). In the beginning of each rainfall event, the infiltration capacity exceeds the rainfall rate for both loam and clay, hence the infiltration rate equals the rainfall rate. When the soil gets saturated the infiltration capacity drops below the rainfall rate and the infiltration rate starts decreasing. Infiltration continues until the rainfall event stops and the ponded water is infiltrated.

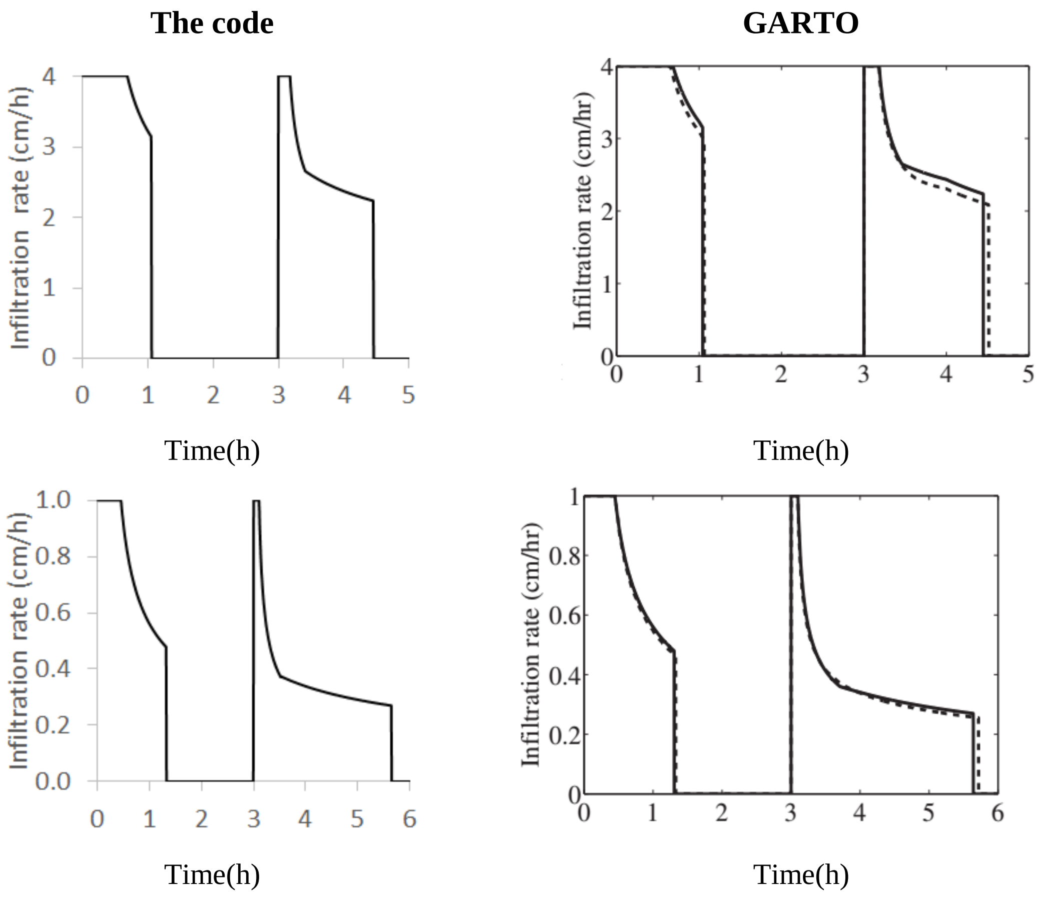


Figure 8.1 Test 1: *Infiltration rate with ponding*
Loam on the top and clay at the bottom. Dotted line is RE FUCG (Lai et al., 2015)

For the first rainfall event, the decreasing part of the infiltration rate curve is smooth for both loam and clay. For the second rainfall event, a sharp edge can be observed at one point for both loam and clay. For loam this sharp point is evident both in the results form Lai et al. (2015) and the code. By analysing the data set, it was found that this point corresponds in time with the merging of two wetting fronts.

Evaluation of Green-Ampt (equation 4) can give a possible explanation of this sharp point. The considered developing wetting front before the merging contains little infiltrated water F. $\theta_0 - \theta_i$, expressed as $\Delta\theta$ is low. Over the timestep where two wetting fronts are merged, θ_i will change to be the soil moisture content of wetting front (*k-2*), not (*k-1*) as before the merge. The value of θ_i decreases. F in wetting front k and F in wetting front (*k-1)* must be summed and considered as F for the merged wetting front. For the initial wetting front, before merging, the capillary forces were a large portion of the downward forces. A larger F and a large $\Delta\theta$ makes the capillary forces small relative to the gravity forces. The infiltration capacity is dependent on the forces that are pulling the wetting front down. The gravity force is constant. The capillary force decreases when the depth and $\Delta\theta$ increases. This makes the change in infiltration capacity decrease when two wetting fronts are merged.

There are two sharp points at the graph for loam in the results from Lai et al. (2015). This is however not the case for the results from the developed code. Lai et al. (2015) do not provide a detailed enough description of their results for further evaluation of this difference.

8.1.2 Wetting front depth

The wetting front depth obtained by the developed code and by Lai et al. (2015) for one hour, three hours and six hours are compared in figure 8.2.

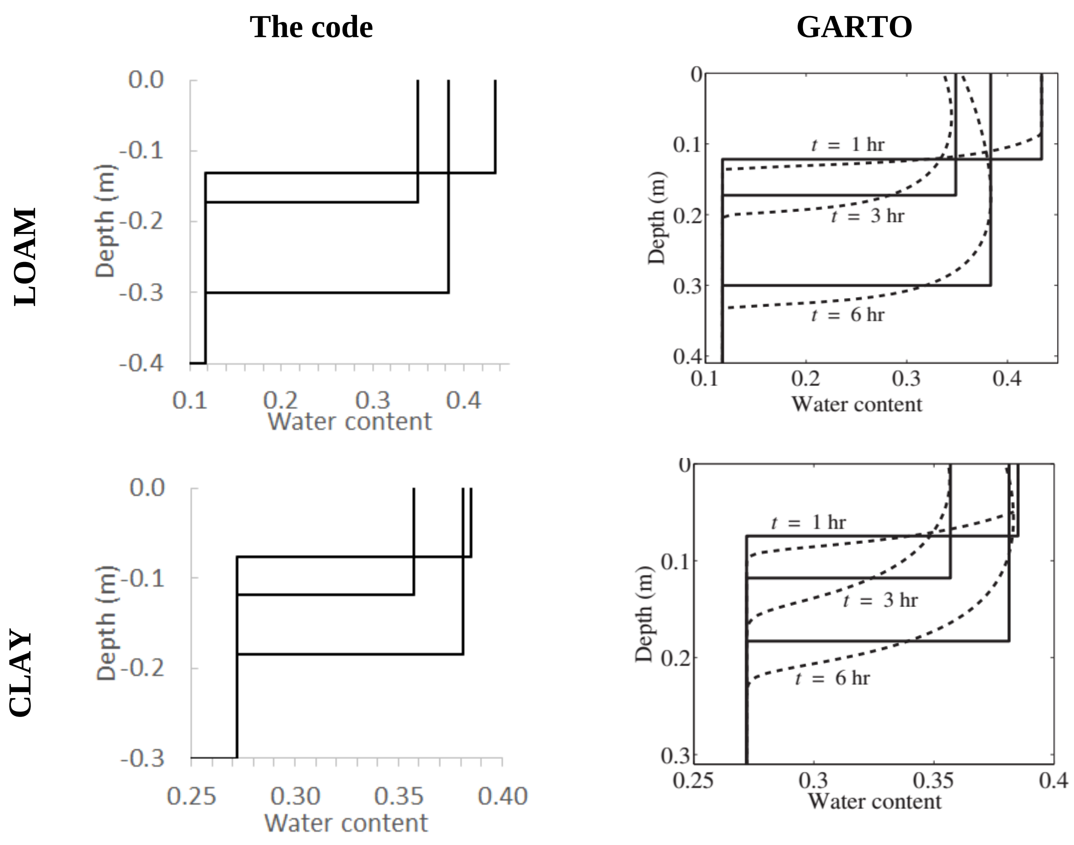


Figure 8.2 Test 1: Wetting front infiltration depth (Lai et al., 2015)

Figure 8.2 indicate that the developed code produces similar wetting front depths as the results by Lai et al. (2015). However, the authors have only presented the graphs of the infiltration depths (as shown in figure 8.2), but not more detailed data. For further evaluating the accuracy of the developed code regarding wetting front depth, more detailed data for comparison would have been needed.

8.1.3 Overall results from Test 1

Table 8.3 presents a comparison of results from Lai et al. (2015) and from the developed code. Time of ponding and deponding, and accumulated infiltrated water depth *F* is presented for loam, clay, and sand.

Table 8.3 *Comparison of results from Lai et al. (2015) and the developed code for loam, clay, sand*

Soil type	Rainfall pulse	Total rain [cm]	Start time [h]	Stopping time [h]	Time ponding [h]		Time deponding [h]		F [cm]	
					GARTO	Result	GARTO	Result	GARTO	Result
Loam	1	4.0	0	1.00	0.686	0.683	1.043	1.042	3.862	3.860
	2	4.0	3	4.00	3.185	3.175	4.442	4.450	2.967	2.945
Clay	1	4.0	0	1.00	0.458	0.450	1.281	1.300	0.851	0.844
	2	4.0	3	4.00	3.105	3.108	5.510	5.633	0.522	0.505
Sand	1	12.5	0	0.25	0.066	0.058	0.318	0.325	10.331	10.219
	2	12.5	3	3.25	3.031	3.025	3.377	3.400	8.916	8.987

Time of ponding, time of deponding, and infiltrated water depths match the results from Lai et al. (2015) with high accuracy (Table 8.3). The largest deviation is on the deponding time for clay after the second rainfall pulse (7,38 minutes). Since the data set only considers two wetting fronts, evaluation of whether there is a cumulative error over time could not be done.

8.1.4 Ponding

In nature, ponding is a delayed process, means ponding will only occur a while into a rainfall event, unless the soil is fully saturated. Ponding will also occur a while after the rainfall event, depending on the soil's infiltration capacity and amount of ponded water. Figure 8.3 shows that results produced by the code corresponds with the nature of ponding.

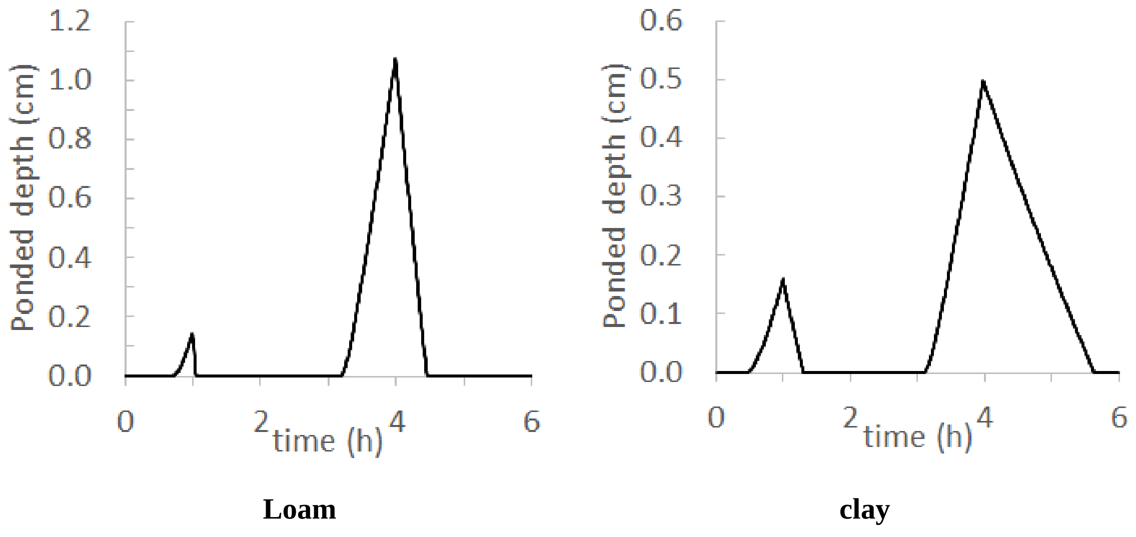


Figure 8.3 Test 1:
Ponded water for clay and loam

For both loam and clay, the two rainfall events start at time zero, and last for one hour. Ponding, however, does start a while into the rainfall event, as shown Figure 8.3 and in Table 8.3. For rainfall event two, the soil has a higher soil moisture content because the water from event one has filled the pores. This leads to decreased infiltration capacity as shown in Figure 8.1, which again leads to more ponding during event two. The deponding time will shift to the right because more water needs to be infiltrated after the rainfall stops, as shown in figure 8.3 and in Table 8.3.

8.1 Test 2

Test 2 represents rainfall sequences with five rainfall pulses all with steady rainfall. As for Test 1, is same rainfall event as used by Lai et al. (2015) was applied on the developed code. The start time (t_i), the duration (d_i) and the rainfall rate (4 cm/h) for each rain pulse are shown in table 8.4. Unlike for test 1, excess rainfall is not allowed to pond but is directly becoming runoff. Excess runoff is this way removed from further infiltration calculations.

Tabell 8.4 *Start of rainfall event (t_i) and duration (d_i)*

Soil	Rainfall rate	t_1	d_1	t_2	d_2	t_3	d_3	t_4	d_4	t_5	d_5
Loam	4	0	2	20	0.5	21	1	40	1	41.5	0.5
Clay	1	0	2	20	0.5	21	1	40	1	41.5	0.5

8.1.1 Infiltration rate

The infiltration rate obtained by the developed code and by Lai et al. (2015) was compared for loam and clay as shown in Figure 8.4 - Figure 8.5

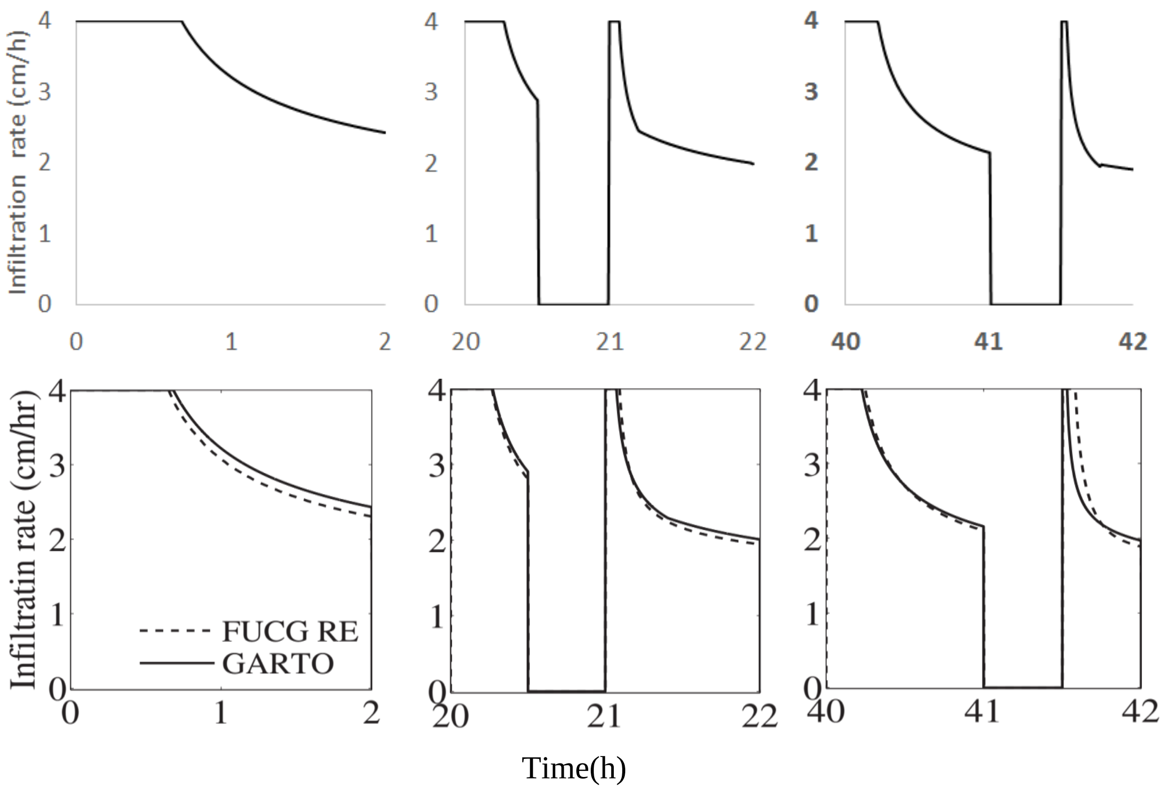


Figure 8.4 Infiltration rate in loam.

Produced results on the top and results from Lai et al. (2015) on the bottom (Lai et al., 2015)

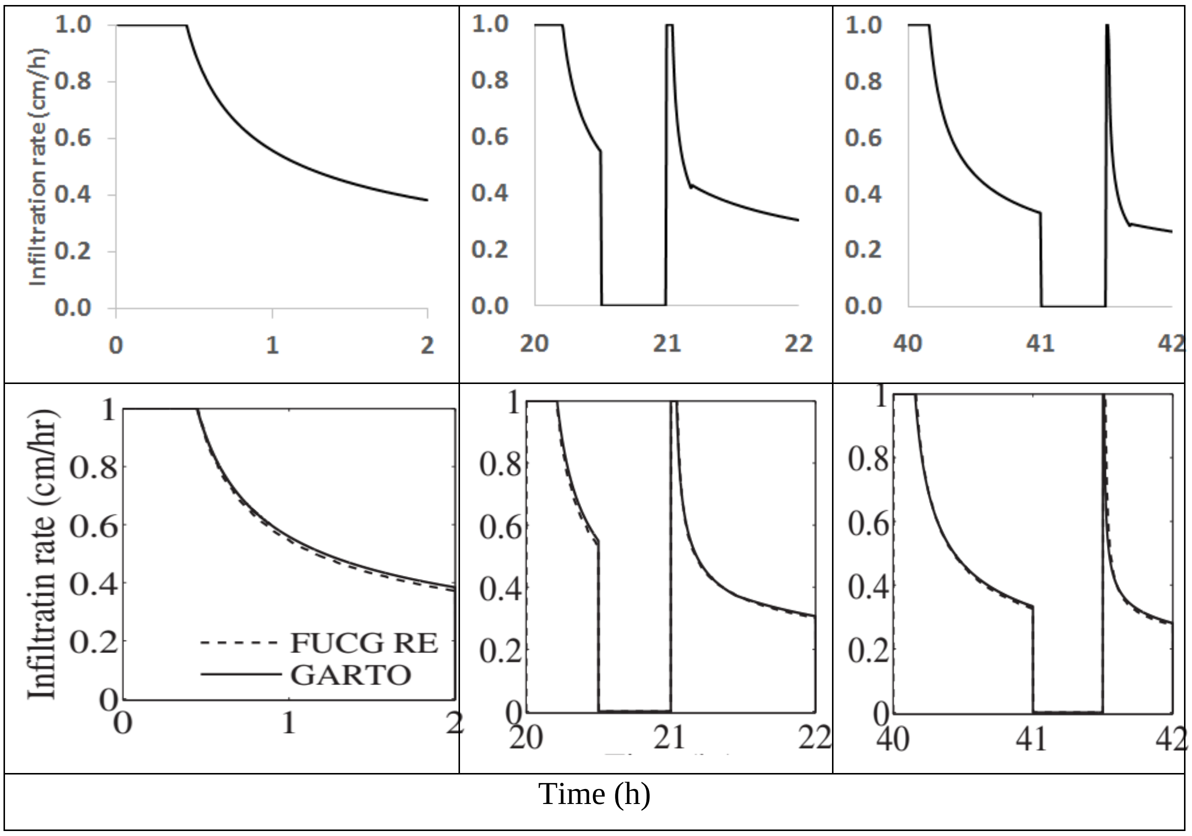


Figure 8.5 Test 2: Infiltration rate in clay.

Produced results on the top and results from Lai et al. (2015) on the bottom (Lai et al., 2015)

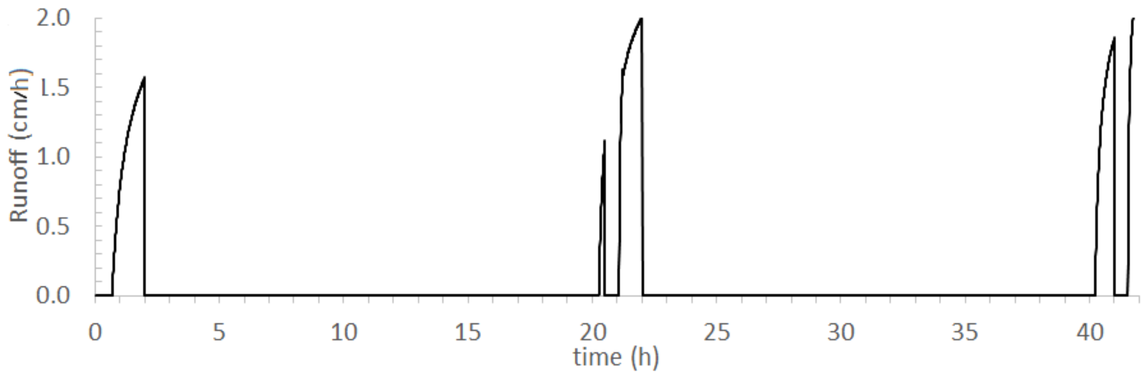
The sharp point discussed for test 1 is also present were two wetting fronts meet in the results produced by the code (Figure 8.4 - Figure 8.5). However, the point cannot be spotted in the graphs provided by Lai at al. (2015)

This might be because no wetting fronts merge during the rainfall event in the study by Lai et al. (2015). This does, however, seem unlikely considering that the results from Lai et al. (2015) and the developed code agree on the merging point in test 1.

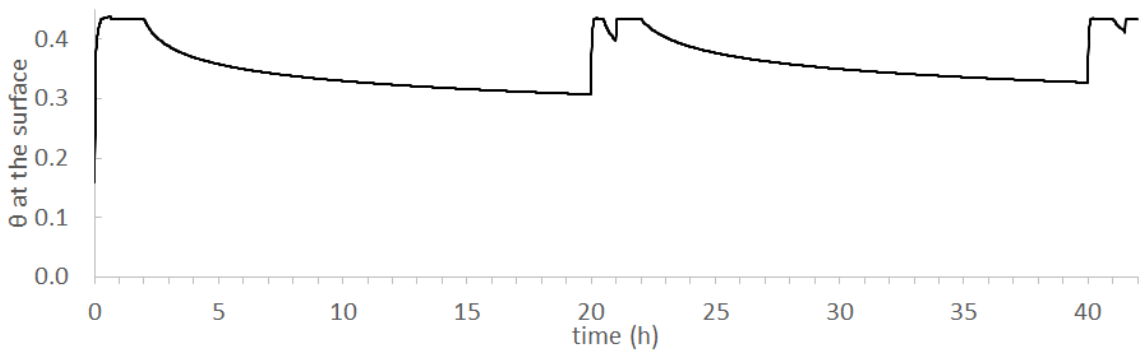
A more likely explanation is that the calculation of infiltration rate over the merging of two wetting fronts is done differently in the developed code and in in the study presented by Lai et al. (2015). The authors do not provide a detailed description of how the infiltration rate changes when two wetting fronts merges. An indication that the calculations might have been conducted differently is the fact that there are discontinuities in the infiltration rate curves in the results from the developed code. Discontinuities in the infiltration rate curve can be seen for both clay and loam. The reason for the discontinuity is not identified. However, investigations on this is recommended as part of further development of the code.

8.1.2 Runoff predictions test 2

In nature, expected runoff will increase with increased soil moisture content. Soil moisture content and corresponding runoff, simulated by the code, is shown in Figure 8.6. The results correspond with the expected natural response.



(a)



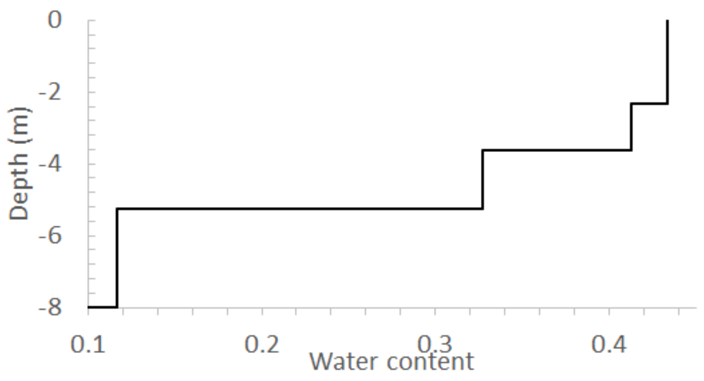
(b)

Figure 8.6 Runoff prediction and soil moisture content for loam test 2

Precipitation event in pulse two and five has the same precipitation rate and duration. However, as seen in Figure 8.6.b, the soil moisture content before pulse two is at around 0.3, while it is at around 0.4 for pulse five. The runoff for pulse two reach a maximum of 1 cm/h while the runoff for pulse five reaches 2 cm/h as shown in Figure 8.6.a. The same results can be found for pulse three and four, but it is not as clear.

8.1.3 Number of wetting fronts

Test 2 with five rainfall pulses will at some stages during the simulations have three wetting fronts in addition to the initial wetting front. Figure 8.7 shows the soil moisture profile at one of these stages.



Figur 8.7 Soil moisture profile for loam after 42,5 hours, test 2

The code does operate with state vectors, holding four elements. Any precipitation series causing more than three wetting fronts plus initial wetting front, would not be possible to run for the code. It would be possible to apply a similar approach as in GAR, where two wetting fronts are merged if the amount of wetting fronts exceeds a certain number (Ogden and Saghafian, 1997). However, the computational costs of having vectors with no limits on number of elements, should not be too high. A natural wetting front would have a curved shape as shown in Figure 3.2. In comparison, Figure 8.7 is a rough reproduction of this curve. An advantage of increasing the number of elements in the vectors would be to produce a smoother curve than in Figure 8.7. This would make the simulations more accurate. Missing programming skills early in the process is the reason for why this not being implemented.

8.2 Comparison against Green Roof

Precipitation data and soil moisture data gathered from a green roof in Trondheim was compared to simulations of the code. The green roof was ten cm. thick with free drainage under the soil. The precipitation rate varied between 0.98 cm/h and 2.7 cm/h and lasted for ten minutes.

Soil properties are shown in table 8.1. Field measurements decided saturated conductivity K_s. Initial soil moisture content θ_i was set equal to the measured soil moisture content in the beginning of the rainfall. Bubbling pressure ψ_b, pore distribution index λ and saturated soil moisture content θ_s were guessed by comparing the soil type of the green roof to the soils presented in Lai et al. (2015).

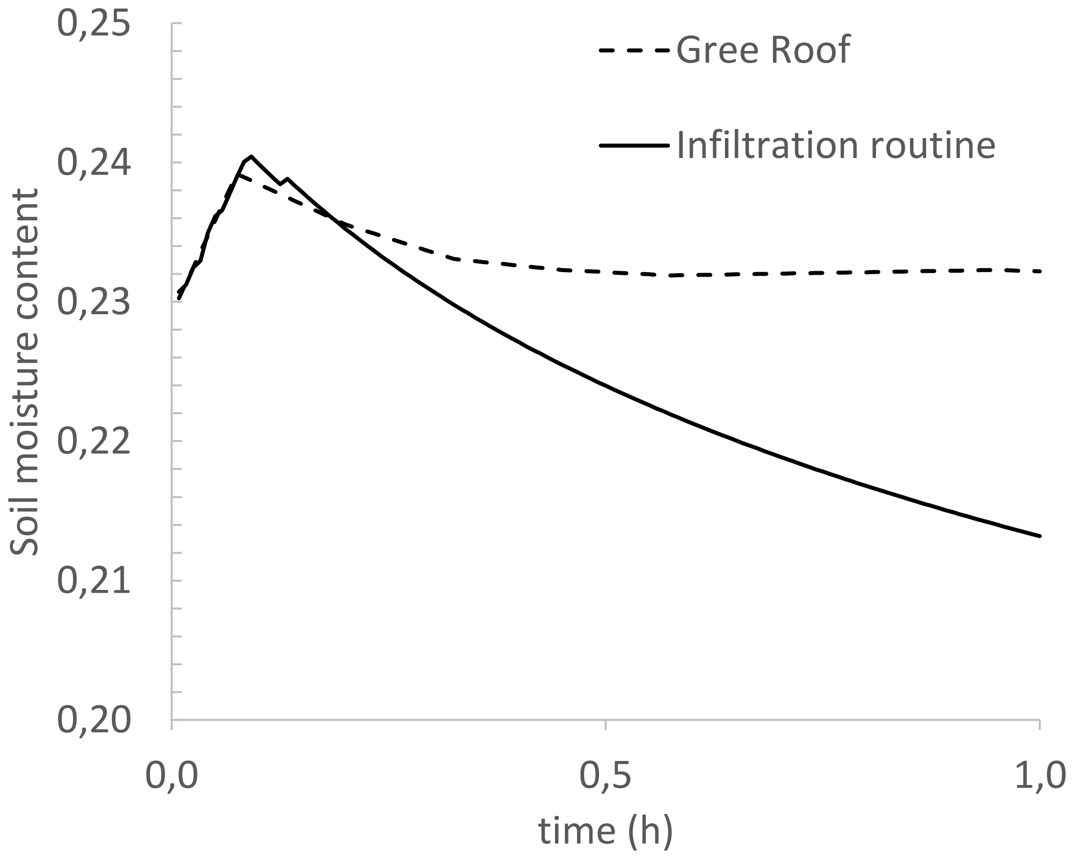


Figure 8.8 Soil moisture content green roof and infiltration routine

The code produces good prediction of the increase in soil moisture content during the rainfall event as shown in figure 8.8. However, during redistribution, the soil moisture content decreases too fast compared to the green roof.

A possible explanation could be connected to the thickness of the roof. The code simulations consider infinitively deep soil columns, while the green roof is only ten cm. thick. When the water in the green roof reaches the bottom of the soil, capillary forces will try to pull the water back up into the soil, like a sponge will hold the water inside. For the soil column, capillary forces will work the opposite way. Below the wetting front, the soil moisture content is less than in the wetting front. Therefore, the dry soil will create capillary forces pulling the water downwards.

For further studies of the code performance on real situations, there should be conducted tests on a deeper soil column, and preferably a denser soil. This would force the soil moisture to move slower through a longer soil column and the simulations would have the same conditions as the test soil for a longer period.

8.1 Sensitivity analysis of parameters

The land use and soil type are important factors for deciding the infiltration capacity. As implied in section 4.2.1, the input parameters for the infiltration routine can be decided either by field measurements or by gathering the parameters from literature data based on the soil type. Uncertainties are present in both these methods.

For field measurements, the main uncertainty is a result of great variation of the surface cover and soil types in urban catchment (Mein and Larson, 1973). Parameters gathered from the literature will in addition have uncertainty connected to misinterpreting of which soil types are present in the evaluated catchment. A brief uncertainty analysis of the five parameters is therefore conducted to investigate the consequence of choosing wrong parameters.

Loam was chosen as test soil and the conditions in test 1 was applied. Each parameter was tested separately while the other parameters remained unchanged. A certain percentage increase and a percentage decrease in value were tested for each parameter. The applied percentage change varied for the parameters. The aim was to reflect the natural variation of the parameter between soil types and at the same time not produce errors that are too significant. Table 8.5 presents the results. The analysis was conducted looking at cumulative infiltrated water depth *F*. The result of a high *F* value would result in low ponding or runoff making *F* a suitably parameter to look at.

Tabell 8.5 Sensitivity analysis on the parameters

Parameter/	Pulse	Decrease		Increase	
Change in %		F [cm]	e (%)	F [cm]	e (%)
K_s	1	3.186	-17.465	4.000	3.627
+20%/-20%	2	1.985	-32.594	3.932	33.510
ψ_b	1	3.460	-10.360	3.998	3.572
+40%/-40%	2	2.469	-16.179	3.321	12.769
λ	1	3.809	-1.328	3.904	1.137
+40%/-40%	2	2.906	-1.328	2.985	1.358
θ_s	1	3.813	-1.211	3.900	1.041
+5%/-5%	2	2.878	-2.279	3.018	2.484
θ_i	1	3.952	2.375	3.715	-3.762
+50%/-50%	2	3.109	5.566	2.778	-5.670
θ_r	1	3.860	0.012	3.860	0.012
+300%/-90%	2	2.958	0.450	2.926	-0.662

The infiltrated water depth *F* is most sensitive to the saturated hydraulic conductivity K_s. Considering that the saturated conductivity varies greatly in nature, a 20% margin of error should be accepted of the user. However, up to 33.5% less water would infiltrate if the K_s is misinterpreted with 20%. On the other hand, *F* is less sensitive to changes in the other variables.

8.2 Steady state rainfall

The infiltration routine was coded with the method of combining GA and Mein and Larson (GAML) (Mein and Larson, 1973). GAML use steady state rainfall, which means that *Pe* is constant throughout the rainfall event. This is rarely the case in nature. A natural rainfall event normally follows a curve with increasing intensity until it reaches a peak, before it drops gradually. The results presented are generated for steady state rainfall events. For a natural rainfall event the code would consider the rainfall intensity for the last timestep as the steady state rainfall, regardless of the intensities of earlier rainfall intensities. Nevertheless, the code managed to predict the increase in soil moisture content in the green roof in section 8.2 even though the rainfall intensity varying.

For further work, two solutions are suggested. Solution (1) is to continuously calculate an average precipitation rate (PeA), including all the previous timesteps in the precipitation event during situation B, Figure 3.1. A suggestion on how the average precipitation rate could be calculated is presented in Equation 18:

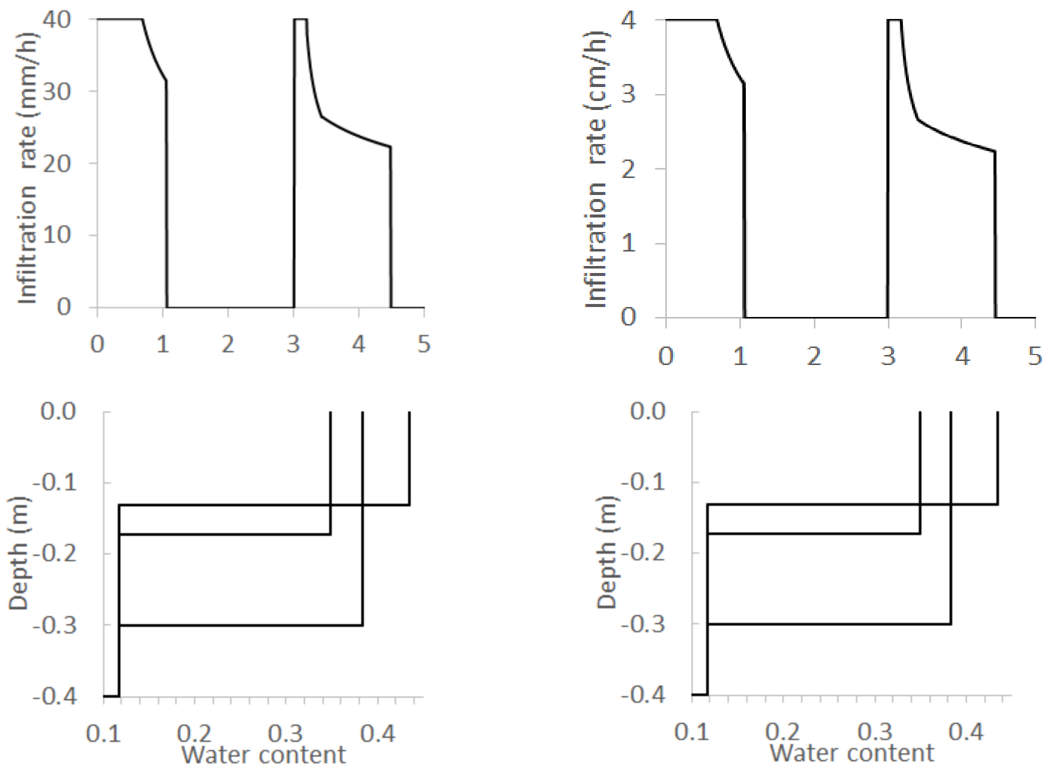
$$\mathrm{PeA} = \frac{\mathrm{PeA} * \left(\frac{\mathrm{time} - \mathrm{deltatime} - \mathrm{timestep}}{\mathrm{timestep}}\right) + \mathrm{Pe}}{\frac{\mathrm{time} - \mathrm{deltatime}}{\mathrm{timestep}}} \tag{18}$$

Where Pe is precipitation rate, deltatime is the time of the beginning of the precipitation event and timestep are the time intervals. This precipitation rate can then be used as a “dynamic” steady rainfall rate to calculate t_p and F_p in each timestep.

Solution (2) is to investigate the modified versions of GAML which allow unsteady rainfall being used (e.g. Chu, 1987). If modifications like solution (1) or (2) are done, the results should be validated against field tests to assess whether it improves the stability of the results.

9 Performance of the infiltration routine in SHyFT

In the following section the results produced in SHyFT is compared to the results produced by the code. The Infiltration curve, water depths, ponding deponding times, and total infiltrated water are compared. The precipitation series presented in table 8.2 and the soil type loam is used for the simulations.



SHyFT **The code**

Figure 9.1 Infiltration curve and infiltrated depth in SHyFT compared to the code

Both the infiltration curve and the infiltrated water depth follows the same pattern in SHyFT and in the code (Figure 9.1). In the code, the unit of length is (cm). During the process, it was discovered that it would have been easier to use the SI unit (mm), so a transition could be preferable for further use of the code. SHyFT, However, operates with (mm) which explains why the unit is different on the infiltration curves. On the depth curves, Both the graphs are presented in the SI unit (m).

Tabell 9.1 Comparison of ponding and deponding time and infiltrated water depth

Soil type	Rainfall pulse	Total rain [cm]	Start time [h]	Stopping time [h]	Time ponding [h]		Time deponding [h]		F [cm]	
					SHyFT	Code	SHyFT	Code	SHyFT	Code
Loam	1	4.0	0	1.00	0.692	0.683	1.042	1.042	3.860	3.860
	2	4.0	3	4.00	3.200	3.175	4.467	4.450	3.089	2.945

Ponding time, deponding time and total infiltrated water depth *F* is predicted similar by SHyFT and the code (Table 9.1). However, SHyFT do calculate the values somewhat different. If this is an error, or if it is due to some minor differences in how the code is implemented is hard to tell. SHyFT is only tested for this one soil type and precipitation series. Further tests are recommended to validate the accuracy of SHyFT. These tests will probably also tell if the differences in table 9.1 is neglectable.

Conclusion

The study was conducted to evaluate the performance of a full hydrological model with an implemented infiltration routine, with the goal of improving hydrological runoff predictions in urban catchments. This was done by developing a code for an infiltration routine in C++. The code was further implemented into the HBV model in SHyFT.

Despite small differences in the shape of the infiltration rate curve when comparing to the results form Lai et al. (2015), the overall performance of the code is good. The comparison against observed data from a green roof showed that increase in soil moisture content is well predicted, while the decrease is overestimated. The runoff and ponding prediction corresponds to the expectation of how these responses behave.

The most sensitive variable was found to be the saturated hydraulic conductivity, K_s. It was further found that choosing a wrong K_s affects the results significantly (20% change in K_s gave up to 33% change in F).

The code is concluded to describe infiltration well. With some further development, the code is promising for implementation and use in the HBV model. Further work is suggested on:

- Implementing a ground water table as explained in the Lai et al. (2015). This would make the code able to predict infiltration for more scenarios.
- Lai et al. (2015) is also working on a layered GARTO model. When this is available, it should be implemented.
- Implementing state vectors handling an indefinite number of wetting fronts.
- Testing SHyFT for more scenarios and debugging any errors that have not yet been discovered.
- Conducting real infiltration measurements to validate the performance of the code, and develop the code to handle errors that might occur.
- Implementing the code fully in the HBV model and test the new version of the HBV model against real catchments. How the runoff from the infiltration routine should be handled in the HBV model needs to be decided. How an urban catchment should be divided into cells should be investigated.

References

AGHAKOUCHAK, A. & HABIB, E. 2010. Application of a Conceptual Hydrologic Model in Teaching Hydrologic Processes. TEMPUS PUBLICATIONS.

BECKER, M. A. 2016. Assesment of downspout disconnecting by modeling infiltration potential in urban areas. *NTNU*.

BROOKS, R. H. & COREY, A. T. 1966. Properties of Porous Media Affecting Fluid Flow. *Journal of the Irrigation and Drainage Division,* Vol. 92 Pg. 61-9.

BUNSRI, T., SIVAKUMAR, M. & HAGARE, D. 2008. Simulation of Water Movement through Unsaturated InfiltrationRedistribution

System *Journal of Applied Fluid Mechanics,* 2.

BUTLER, D. & DAVIS, J. W. 2004. Urban Drainage, second edition.

CHU, S. T. 1987. Generalized Mein&#x2010;Larson Infiltration Model. *Journal of Irrigation and Drainage Engineering,* 113**,** 155-162.

CORRADINI, C., MELONE, F. & SMITH, R. E. 1997. A unified model for infiltration and redistribution during complex rainfall patterns. *Journal of hydrology,* 192.

DINGMAN, L. D. 2015. Physical Hydrology.

GOWDISH, L. & MUÑOZ-CARPENA, R. 2009. An Improved Green–Ampt Infiltration and Redistribution Method for Uneven Multistorm Series All rights reserved. No part of this periodical may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or any information storage and retrieval system, without permission in writing from the publisher. *Vadose Zone Journal,* 8**,** 470-479.

GREEN, W. H. & AMPT, G. A. 1911. Studies on Soil Phyics. *The Journal of Agricultural Science,* 4**,** 1-24.

HELSET, S. 2017. Statkraft. Available: https://github.com/statkraft/shyft/wiki/BuildCplusplus [Accessed 28.05 2017].

HORTON, R. E. 1933. The Rôle of infiltration in the hydrologic cycle. *Eos, Transactions American Geophysical Union,* 14**,** 446-460.

HUNDECHA, Y. & BÁRDOSSY, A. 2004. Modeling of the effect of land use changes on the runoff generation of a river basin through parameter regionalization of a watershed model. *Journal of Hydrology,* 292**,** 281-295.

JACOBSON, C. R. 2011. Identification and quantification of the hydrological impacts of imperviousness in urban catchments: A review. *Journal of Environmental Management,* 92**,** 1438-1448.

KANG, I. S., PARK, J. I. & SINGH, V. P. 1998. Effect of urbanization on runoff characteristics of the On-Cheon Stream watershed in Pusan, Korea. *Hydrological Processes,* 12**,** 351-363.

KIRKLAND, M. R., HILLS, R. G. & WIERENGA, P. J. 1992. Algorithms for solving Richards' equation for variably saturated soils. *Water Resources Research,* 28**,** 2049-2058.

LAI, W., OGDEN, F. L., STEINKE, R. C. & TALBOT, C. A. 2015. An efficient and guaranteed stable numerical method for continuous modeling of infiltration and redistribution with a shallow dynamic water table. *Water Resources Research,* 51**,** 1514-1528.

LINDSTRÖM, G., JOHANSSON, B., PERSSON, M., GARDELIN, M. & BERGSTRÖM, S. 1997. Development and test of the distributed HBV-96 hydrological model. *Journal of Hydrology,* 201**,** 272-288.

MEIN, R. G. & LARSON, C. L. 1973. Modeling infiltration during a steady rain. *Water Resources Research,* 9**,** 384-394.

NKSS 2015. Klima i Norge 2100.

NYHUS, E. 2016. Infiltrasjonsrutine i en full hydrologisk modell.

OGDEN, F. L. & SAGHAFIAN, B. 1997. Green and Ampt Infiltration with Redistribution. *Journal of Irrigation and Drainage Engineering,* 123**,** 386-393.

RICHARDS, L. A. 1931. CAPILLARY CONDUCTION OF LIQUIDS THROUGH POROUS MEDIUMS. *Physics,* 1**,** 318-333.

RODGERS, J. D. & HASSELMANN, K. F. 1997. Evolving laws and policies for an ever-present risk.

ROSS, P. J. 1990. Efficient numerical methods for infiltration using Richards' equation. *Water Resources Research,* 26**,** 279-290.

SALVADORE, E., BRONDERS, J. & BATELAAN, O. 2015. Hydrological modelling of urbanized catchments: A review and future directions. *Journal of Hydrology,* 529, Part 1**,** 62-81.

SAVITCH, W. 2010. Absolute C++. fourth edition.

SINGH, V. P. 1995. *Computer models of watershed hydrology,* Colorado, Water Resources Publications.

TALBOT, C. A. & OGDEN, F. L. 2008. A method for computing infiltration and redistribution in a discretized moisture content domain. *Water Resources Research,* 44**,** n/a-n/a.

Appendix A: User manual

To run the code for a new soil type. Follow the steps below

1) Open the main.cpp file. This will open the entire project. CodeBlocks is used to develop the program
2) Go to the header-file “Parameters.h”. In this file, all the soil specific parameters and properties are listed. Change them to the wanted soil properties.
3) Change the ponding conditions to “Yes” if runoff water can pond, or “No” if it cannot.
4) Change the name to the name of the file where the precipitation data is stored (default “Precipitaion.txt”). This file needs to be in the same folder as all the other project files.
5) Changes in the depth of initial soil column can be done in main where the depth is initialized.
6) Create a precipitation series. Go to the file specified as the precipitation file. It is easiest to make a precipitation series in excel and copy-paste this series into the precipitation-file. The file needs to have space-separated precipitation data and points are used as decimal separator.
7) Build and run the code. A window will open and display output data specified in the end of “main”.
8) Use the excel-sheet analyse data Master to evaluate the full result. Open the document, go to data, click update data and import. The new data will automatically be imported to the excel document form the file “Output.txt”.

Appendix B: The code

Main

```
#include <iostream>
#include <fstream>
#include <cmath>
#include <windows.h>
//Includes the classes
#include "soilMoisture.h"
#include "getPrecipitation.h"
#include "Parameters.h"
#include "Infiltration.h"
#include "Infiltration_depth.h"
#include "Ponding.h"

using namespace std;

int main() {
	//Allows the program to print to the file "Output", and sets print precision = 3 
decimals
	ofstream theFile("Output.txt");
	std::cout.precision(3);
	//Allows the program to enter classes, and name class objects
	getPrecipitation number;
	Parameters Par;
	Infiltration Ff;
	soilMoisture output;
	Infiltration_depth getZ;
	Ponding Pond;
	//Runs through the precipitation file and reads the length of file
	number.setLength(); int length = number.getLength();
	//Stores the Pe array and reads from file Precipitation
	char filename[50];	//initialize a variable of type char to store the filename
	filename[50] = Par.getfilename(filename);  //The function "getfilename" returns the 
file name from "Parameters" and Stores the file name as a char "filename"
	ifstream fp;			//Makes an input file stream class
	fp.open(filename);		//Opens the file named in the variable "filename"
	float Prec;
	float Pe[length];		//Creates a precipitation vector with the length "length"
	int j = 0;
	while (fp.good()) {	//While-loop runs through the file "filename" until the end
		fp >> Prec;		//Each data is stored as "Prec"
		Pe[j] = Prec;		//Stores Precipitation value in a vector
		j++;			//increase the value of j with one
	}
	fp.close();			//Closes the file "filename"
						//INITIALIZTION;
	float ORi = Par.getORi(), Hc = Par.getHc(), Yb = Par.getYb(), lambda = 
Par.getlambda(), timestep = Par.getTimestep(); //Gets parameters from the class "Parame-
ters"
```

```
float time = 0;
	float Oi = Par.getOi(), Os = Par.getOs(), Ks = Par.getKs(), Ki = Par.getKi(), Or
= Par.getOr(); //Gets parameters from the class "Parameters"
	float Freal = 0;
	float deltaF = 0;
	float f = 0;
	float ftest = 0;
	float O0[4] = { 0 }; O0[0] = Oi;
	float OR0[4] = { 0 }; OR0[0] = ORi;
	float K0[4] = { 0 }; K0[0] = Ki;
	float Z[4] = { 0 }; Z[0] = 10;
	int k = 0;
	float ponding = 0;
	float Runoff = 0;
	float PeL = 0;
	Par.geterror();  //Checks if it is an error in the input values in "Parameters"
	//STARTS THE FOR LOOP
	for (int i = 0; i < length; i++) { runs through each precipitation data
		//If I want to slow down the output
		Sleep(0);
		// changes the time in steps of timestep
		time = time + timestep;     //Increases time with one timestep
		//changes k if a new wetting front is formed
		if (Pe[i] > Ks && PeL < Ks && O0[k] < Os) {
			k++;   //Number of wetting fronts "k" is increased by one
		}
//Calculates advance and controls the water demand for wetting front (1 to k-1)
		//Creates a vector controlling water demand for wetting front 1-(k-1)
		float Pek[4] = { 0 };
		//Creates a vector controlling volume for wetting front 1-(k-1)
		float Vzk[4] = { 0 };
		float PeR = Pe[i];          //Creates a variable for residual rainfall
		for (int j = 1; j < k; j++) {          //For wetting front 1-(k-1)
			Vzk[j] = Z[j] * (O0[j] - O0[j - 1]);  //Check initial water volume
			getZ.setZKn(K0[j], K0[j - 1], O0[j], O0[j - 1], Z[j], OR0[j],
OR0[j - 1]);  //Get new depth value
			Z[j] = getZ.getZKn();
			//Water demand is new volume - initial volume Vzk
			Pek[j] = (Z[j] * (O0[j] - O0[j - 1]) - Vzk[j]);
			//Residua rainfall = rainfall - water demand all wetting fronts
			PeR = PeR - Pek[j] / timestep;
		}
//Gets "f", "ftest", "Freal", "deltaF", and "error" form the class "Infiltration"
		Ff.setFf(time, Pe[i], ponding, k, O0[k - 1], i);
		f = Ff.getf();
		ftest = Ff.getftest();
		Freal = Ff.getF();
		deltaF = Ff.getdeltaF(k);
//Considers if the area have free surface runoff or if Ponding occurs
		//Calculates "ponding" and "Runoff" from the class "Ponding"
		Pond.setPonding(f, Pe[i], timestep, i);
		ponding = Pond.getPonding();
		Runoff = Pond.getRunoff();
```

```
//New......................................................................
              //If "Pe" > Ks and a new wetting front is formed --> situation New
              if (Pe[i] > Ks && PeL < Ks && O0[k] < Os) {
                     getZ.setZK0(O0[k - 1], OR0[k - 1]);  //new depth equals dry depth
                     Z[k] = getZ.getZK0();
                     if (Pe[i] > ftest) {   //"Pe" is higher than infiltration capacity
                            O0[k] = Os;     //the soil is saturated
                     }
                     else { //else get soil moisture content
                            output.setO0(Z[k], PeR, O0[k - 1], OR0[k - 1]);
                            O0[k] = output.getO0();
                     }
              }
              else {
//C........................................................................
                     if (Pe[i] >= ftest) { //if "Pe" higher than infiltration capacity
                            getZ.setZs(O0[k - 1], Z[k]);    //get new depth
                            Z[k] = getZ.getZs();
                            O0[k] = Os;                     //The soil is saturated
                     }
//B........................................................................
                     float O0test = O0[k];       //Creates a test variable "O0test"
                     float Ztest = Z[k];              //Creates a test variable "Ztest"
                     if (Pe[i] < ftest && Pe[i] >= Ks) {   //If Pe > ks but less than
                            //Creates a test variable "Ftest"
                            float Ftest = Freal - deltaF - f*timestep;
                            //Runs if "Ftest" is less than the infiltrated water volume
                            while (Ftest < Freal - deltaF) {
                                   //increase "O0test" in small steps
                                   O0test = O0test + 0.0001;
                                   getZ.setZK1(K0[k], O0test, O0[k - 1], Z[k], OR0[k],
OR0[k - 1]); //Get new depth with new O0test as input
                                   Ztest = getZ.getZK1();
                                   //Calculate new "Ftest" volume
                                   Ftest = Ztest*(O0test - O0[k - 1]);
                                          }
                            Z[k] = Ztest;   //New depth is set to "Ztest"
                            O0[k] = O0test; //New soil moisture content = "O0test"
                     }
//A........................................................................
                     if (Pe[i] < Ks) {                 //If "Pe" < "Ks" --> Situation A
                            if (ponding > 0) {         //If still ponding --> situation
C
                                   getZ.setZs(O0[k - 1], Z[k]);   //New depth
                                   Z[k] = getZ.getZs();
                                   O0[k] = Os;          //The soil is fully saturated
                            }
                            else {
                                   //Creates a test variable "Ftest"
                                   float Ftest = Freal - deltaF - 0.001;
                                   //Runs if "Ftest" is less than infiltrated water
                            volume
                                   while (Ftest < Freal - deltaF) {
                                          //increase "Ztest" in small steps
                                          Ztest = Ztest + 0.001;
                                          //Get new "O0" with new O0test as input
                                          output.setO0(Ztest, PeR, O0[k], OR0[k - 1]);
                                          O0test = output.getO0();
                                          //Calculate new "Ftest" volume
                                          Ftest = Ztest*(O0test - O0[k - 1]);
                                   }
                                   Z[k] = Ztest;     //New depth is set to "Ztest"
```

```
                                    Z[k] = Ztest;     //New depth is set to "Ztest"
                                    //New soil moisture content is set to "O0test"
                                    O0[k] = O0test;
                             }
                      }
//.........merges to wetting fronts if they reach the same level..................
                      if (Z[k] > Z[k - 1] && k > 0) {
                             Z[k - 1] = ((O0[k] - O0[k - 1])*Z[k] + (O0[k - 1] - O0[k -
2])*Z[k - 1]) / (O0[k] - O0[k - 2]); //New depth of both wetting fronts
                             //New soil moisture content of both wetting fronts
                             O0[k - 1] = O0[k];
                             //deletes last wetting front because of merging to one
                             Z[k] = 0, O0[k] = 0, K0[k] = 0, OR0[k] = 0;
                             k--;      //Number of wetting fronts "k" is reduced by one
                      }
               }
//OR0 is set using the new calculated soil moisture content
               OR0[k] = (O0[k] - Or) / (Os - Or);
//New K0 by using new calculated relative volumetric soil moisture content
               K0[k] = Ks*pow(OR0[k], 3 + 2 / lambda);
               //PRINTS TO SCREEN
               std::cout << "time " << time;
               std::cout << ": Pe " << Pe[i];
               std::cout << ": k " << k;
               std::cout << ": Z[1] " << Z[1];
               std::cout << ": O0 " << O0[k];
               //PRINTS TO FILE
               std::cout << std::fixed << endl;
               theFile << time << " " << Pe[i] << " " << Freal << " " << f << " " <<
ftest << " " << Z[0] << " " << Z[1] << " " << Z[2] << " " << Z[3] << " " << O0[0] << "
" << O0[1] << " " << O0[2] << " " << O0[3] << " " << OR0[0] << " " << OR0[1] << " " <<
OR0[2] << " " << OR0[3] << " " << K0[0] << " " << K0[1] << " " << K0[2] << " " <<
K0[3] << " " << Pek[0] << " " << Pek[1] << " " << Pek[2] << " " << ponding << " " <<
Runoff << " " << endl;
       //Stores last Precipitation for use to check if new wetting front is formed
               PeL = Pe[i];
       }
       //PRINTS INITIAL VALUES TO SCREEN
       cout << endl << "length of file = " << length << endl << "time = " << time <<
endl << "timestep = " << timestep << endl << "Ks = " << Ks << endl << "Ki = " << Ki <<
endl << "Hc = " << Hc << endl << "Yb = " << Yb << endl << "lambda = " << lambda <<
endl << "Os = " << Os << endl << "Oi = " << Oi << endl << "Or = " << Or << endl <<
"ORi = " << ORi << endl << endl;
       theFile.close();                    //Closes the output file
}
```

Infiltration

Infiltration.h

```
#ifndef INFILTRATION_H
#define INFILTRATION_H
class Infiltration
{
public:
	float setFf(float time, float Pe, float ponding, int k, float O0L_, int i);
	float getf();
	float getF();
	float getftest();
	float getdeltaF(int k);

private:
	float F = 0;
	float f;			//Actual infiltration rate at all times
	float ftest;		//Potential infiltration rate at all times
	float deltatime = 0;	//The time when last wetting front started
	float deltaF[4] = { 0 };  //A vector that stores infiltrated water for wetting 
front 0-k
	float ktest = 0;		//Test variable to check if a new wetting front is created
	float Ftest1;		//
	float Ftest2;
	float Fpot;
	float O0L;
};
#endif // INFILTRATION_
```

Infiltration.cpp

```
#include <iostream>
#include <cmath>
#include "Infiltration.h"
#include "Parameters.h"

using namespace std;

Parameters param;

float Infiltration::setFf(float time, float Pe, float ponding, int k, float O0L_, int i) {
	//Gathers parameters from the class "Parameters"
	float Hc = param.getHc();
	float Ks = param.getKs();
	float Os = param.getOs();
	float timestep = param.getTimestep();

	float Fp;
	float tp;
	float t = 0;
	float Ftest;

	if (i == 0) {
		O0L = O0L_;
	}
	//Calculates infiltration capacity if the precipitation rate would be high enough
	if (k > ktest) {
		deltatime = time;
		deltaF[k] = F;
		ktest = k;
		O0L = O0L_;
	}
	else if (k < ktest) {
		ktest = k;
		O0L = O0L_;
	}
	Ftest1 = F - deltaF[k] + Pe*timestep;
	ftest = Ks*((Hc*(Os - O0L)) / Ftest1 + 1);

	Fp = (Hc*Ks*(Os - O0L)) / (Pe - Ks);
	tp = Fp / Pe;
	Ftest = F - deltaF[k];

	if (Pe > 0 && Pe < Ks) {
		F = F + Pe*timestep;
		f = Pe;
	}
	else if (tp < timestep && Pe > ftest) {
		//iterates Fpot against Ftest similar to the way they do it in GA
		while (Fpot < (Ftest2 - 0.0001)) {
			if (Fpot < Ftest2) {
				// adds a small sum to Fpot until Fpot is 0,001 from Ftest
				Fpot = Fpot + 0.00011;
				Ftest2 = Ks*(time - deltatime + timestep) + Hc*(Os -
O0L)*log(1 + (Fpot / (Hc*(Os - O0L))));
			}
		}
```

```
			F = F + Ftest2;
			f = Ks*((Hc*(Os - O0L)) / (F - deltaF[k]) + 1);
		}
		else if (Pe > Ks) {
			while (t < time - deltatime + timestep) {
				Ftest = Ftest + 0.0001;
				t = tp + (1 / Ks)*(Ftest - Fp + Hc*(Os - O0L)*log((Hc*(Os - O0L) +
Fp) / (Hc*(Os - O0L) + Ftest)));
			}
			if (t <= tp) {
				F = F + Pe*timestep;
				f = Pe;
			}
			else {
				F = F + f*timestep;
				f = Ks*((Hc*(Os - O0L)) / (F - deltaF[k]) + 1);
			}
		}
		else if (ponding > 0) {
			F = F + f*timestep;
			f = Ks*((Hc*(Os - O0L)) / (F - deltaF[k]) + 1);
		}
		else {
			f = 0;
		}
}
float Infiltration::getF() {
		return F;
}
float Infiltration::getf() {
		return f;
}
float Infiltration::getftest() {
		return ftest;
}
float Infiltration::getdeltaF(int k) {
		return deltaF[k];
}
```

getPrecipitation

getPrecipitation.h

```
#ifndef PRECIPITATIONDATA_H
#define PRECIPITATIONDATA_H


class getPrecipitation
{
public:
	float setLength();	//Calculates the length of the precipitation file
	int getLength();	//returns the length


private:
	int length;		//Stores the length
};

#endif // PRECIPITATIONDATA_H
```

getPrecipitation.cpp

```
#include <fstream>
#include <iostream>
#include <cstdlib>
#include "Parameters.h"
#include "getPrecipitation.h"

using namespace std;

Parameters parame;

float getPrecipitation::setLength() {
	char filename[50];         //Creates a variable "filename" of type char
	filename[50] = parame.getfilename(filename);  //Get filename from "Parameters" and 
Stores the file name in "filename"
	ifstream fp;               //Makes an input file stream class called fp
	fp.open(filename);         //Opens the filename given to the variable filename
	if (!fp.is_open()) {       //If file was not properly opened
		cout << "ERROR: The typed file does not exist in the right folder" << endl; 
//Error message returned if the file was not opened
		exit(EXIT_FAILURE); //Exits the program if the file was not properly opended
	}
	int x = 0;                 //Initialize the a counting variable "x" to zero
	float Prec;                //Initialize a variable "Prec" to receive data from fp
	float Ks = parame.getKs();
	while (fp.good())          //Runs as long as there is data in
	{
		fp >> Prec;
		if (Prec > 0 && Prec < 0.00000001) {     //if "Prec" has to many decimals
			cout << "ERROR: Precipitation data at time " << (x + 1)*0.008333 << 
"h has more than 8 decimals" << endl; //Error message returned
			exit(EXIT_FAILURE);                   //Exit the program
		}if (Prec < 0) {                          //if "Prec" has a negative value
			cout << "ERROR: Precipitation data at time " << (x + 1)*0.008333 << 
"h is < 0" << endl; //Error message returned
			exit(EXIT_FAILURE);        //Exit the program
		}
		x++;                  //counts the number of data
	}
	length = x;                //store the length of the file in the variable "Length"
	fp.close();                //closes the file
}
int getPrecipitation::getLength() { //returns the length to main
	return length;
}
```

Infiltration_depth

Infiltration_depth.h

```
#ifndef INFILTRATION_DEPTH_H
#define INFILTRATION_DEPTH_H


class Infiltration_depth
{
public:
	float setZKn(float K0, float K0L, float O0, float O0L, float Z_, float OR0, float
OR0L);
	float setZK1(float K0, float O0, float O0L, float Z_, float OR0, float OR0L);
	float setZK0(float O0L, float OR0L);
	float setZs(float O0L, float Z_);
	float getZKn();
	float getZK1();
	float getZK0();
	float getZs();

private:
	float ZK1;
	float ZKn;
	float ZK0;
	float Zs;
};

#endif // INFILTRATION_DEPTH_H
```

Infiltration_depth.cpp

```
#include <iostream>
#include <cmath>
#include "Infiltration_depth.h"
#include "Parameters.h"
using namespace std;
//Gathers parameters from the class "Parameters"
Parameters par; //Gathers the parameters from the class "Parameters"
float Yb = par.getYb();
float Os = par.getOs();
float lambda = par.getlambda();
float ORi = par.getORi();
float Ks = par.getKs();
float Hc = par.getHc();
float timestep = par.getTimestep();
float k1, k2, k3, k4;
//Calculates the advance in Z direction for a wetting front k > 1
float Infiltration_depth::setZKn(float K0, float K0L, float O0, float O0L, float Z_, float
OR0, float OR0L) {
	float G;	//initialize G
	if (O0 < Os) {	//Unsaturated G
		G = Yb*(pow(OR0, 3 + 1 / lambda) - pow(OR0L, 3 + 1 / lambda)) / (3 * lambda
+ 1);
	}
	else {		//Saturated G
		G = Yb*(3 * lambda + 2 - pow(OR0L, 3 + 1 / lambda)) / (3 * lambda + 1);
	}		//RK4 calculations of k1-k4
	k1 = timestep*((K0 - K0L) / (O0 - O0L))*((G / Z_) + 1);
	k2 = timestep*((K0 - K0L) / (O0 - O0L))*((G / (Z_ + k1*0.5)) + 1);
	k3 = timestep*((K0 - K0L) / (O0 - O0L))*((G / (Z_ + k2*0.5)) + 1);
	k4 = timestep*((K0 - K0L) / (O0 - O0L))*((G / (Z_ + k3)) + 1);
	ZKn = Z_ + (k1 + 2 * k2 + 2 * k3 + k4) / 6;	//Adds delta Z to Z
}
//Infliltration Depth for wettingfront k = 1;
float Infiltration_depth::setZK1(float K0, float O0, float O0L, float Z_, float OR0, float
OR0L) {
	float G;
	if (O0 < Os) {
		G = Yb*(pow(OR0, 3 + 1 / lambda) - pow(OR0L, 3 + 1 / lambda)) / (3 * lambda
+ 1);
	}
	else {
		G = Yb*(3 * lambda + 2 - pow(OR0L, 3 + 1 / lambda)) / (3 * lambda + 1);
	}		//RK4 calculations of k1-k4
	k1 = timestep*((K0 / (O0 - O0L)) + (Ks*G / ((O0 - O0L)*Z_)));
	k2 = timestep*((K0 / (O0 - O0L)) + (Ks*G / ((O0 - O0L)*(Z_ + k1*0.5))));
	k3 = timestep*((K0 / (O0 - O0L)) + (Ks*G / ((O0 - O0L)*(Z_ + k2*0.5))));
	k4 = timestep*((K0 / (O0 - O0L)) + (Ks*G / ((O0 - O0L)*(Z_ + k3))));
	ZK1 = Z_ + (k1 + 2 * k2 + 2 * k3 + k4) / 6;	//Adds delta Z to Z
}
//Infliltration Depth for a saturated wettingfront;
float Infiltration_depth::setZs(float O0L, float Z_) {
	//RK4 calculations of k1-k4
	k1 = timestep*(Ks / ((Os - O0L))*(Hc / Z_ + 1));
	k2 = timestep*((Ks / (Os - O0L))*(Hc / (Z_ + k1*0.5) + 1));
	k3 = timestep*((Ks / (Os - O0L))*(Hc / (Z_ + k2*0.5) + 1));
	k4 = timestep*((Ks / (Os - O0L))*(Hc / (Z_ + k3) + 1));
	Zs = Z_ + (k1 + 2 * k2 + 2 * k3 + k4) / 6;	//Adds delta Z to Z
}
```

```
//Infiltration Depth for first advance for a new wetting front;
float Infiltration_depth::setZK0(float O0L, float OR0L) {
	float tau = (timestep*Ks / (Os - O0L));
	float G = (Yb / lambda)*((3 * lambda + 2) - pow(OR0L, (3 + (1 / lambda)))) / (3 +
(1 / lambda)); // equation 13 GARTO
	ZK0 = 0.5*(tau + sqrt(pow(tau, 2) + 4 * tau*G));
}
float Infiltration_depth::getZKn() {
	return ZKn;
}
float Infiltration_depth::getZK1() {
	return ZK1;
}
float Infiltration_depth::getZK0() {
	return ZK0;
}
float Infiltration_depth::getZs() {
	return Zs;
}
```

Parameters

Parameters.h

```
#ifndef INFILTRATION_DEPTH_H
#define INFILTRATION_DEPTH_H


class Infiltration_depth
{
public:
	float setZKn(float K0, float K0L, float O0, float O0L, float Z_, float OR0, float
OR0L);
	float setZK1(float K0, float O0, float O0L, float Z_, float OR0, float OR0L);
	float setZK0(float O0L, float OR0L);
	float setZs(float O0L, float Z_);
	float getZKn();
	float getZK1();
	float getZK0();
	float getZs();

private:
	float ZK1;
	float ZKn;
	float ZK0;
	float Zs;
};

#endif // INFILTRATION_DEPTH_H
```

Parameters.cpp

```
#include <iostream>
#include <cmath>
#include <string>
#include <cstdlib>
#include "Parameters.h"

using namespace std;

float Parameters::getKs() {
	return Ks;
}
float Parameters::getYb() {
	return Yb;
}
float Parameters::getlambda() {
	return lambda;
}
float Parameters::getOs() {
	return Os;
}
float Parameters::getOi() {
	return Oi;
}
float Parameters::getOr() {
	return Or;
}
float Parameters::getORi() {
	ORi = (Oi - Or) / (Os - Or);        //Relative initial volumetric watercontent
	return ORi;
}
float Parameters::getKi() {
	Ki = Ks*pow(ORi, 3 + 2 / lambda); //initial soilmoisture conductivity
	return Ki;
}
float Parameters::getHc() {
	Hc = Yb*((2 + 3 * lambda) / (1 + 3 * lambda)); //initial soilmoisture conductivity
	return Hc;
}
float Parameters::getTimestep() {
	float timestep = minutes / 60;
	return timestep;
}
char Parameters::getPond(char* outPond) { //gathers if ponding and stores it in a string
	char str[10];
	for (int i = 0; i < 10; ++i) {
		outPond[i] = Pond[i];
	}
}
char Parameters::getfilename(char* outFilename) { //gathers filename and stores it in a
string
	char str[50];
	for (int i = 0; i < 50; ++i) {
		outFilename[i] = filename[i];
	}
}
```

```
float Parameters::geterror() {
	float error;
	if (Ks<0 || Yb<0 || lambda<0 || Os<0 || Oi<0 || Or<0 || ORi<0 || Ki<0 || Hc<0 ||
minutes<0) {
		cout << "ERROR: A Parameter in the class Parameters is < 0" << endl; //Error
message returned
		exit(EXIT_FAILURE);			//Exit the program
	}
	else if (Or > Oi || Oi > Os || Or > Os) {
		cout << "EROOR: The variable Or > Oi or Oi > Os or Or < Os" << endl; //Error
message returned
		exit(EXIT_FAILURE);
	}
	else if (Or >= 1 || Oi >= 1 || Os >= 1) {
		cout << "ERROR: Or or Oi or Os >= 1" << endl; //Error message returned
		exit(EXIT_FAILURE);
	}
	return error;
}
```

Ponding

Ponding.h

```
#ifndef PONDING_H
#define PONDING_H


class Ponding
{
public:
	float setPonding(float ftest, float Pe, float timestep, int i);
	float getPonding();
	float getRunoff();

private:
	float ponding = 0;
	float Runoff = 0;
};

#endif // PONDING_H
```

Ponding.cpp

```cpp
#include "Ponding.h"
#include <iostream>
#include <cstring>
#include <string>
#include <cstdlib>
#include "Parameters.h"


using namespace std;

Parameters paramet;

float Ponding::setPonding(float f, float Pe, float timestep, int i) {
	char En[10];
	En[10] = paramet.getPond(En); //returns ponding condition from parameters
	if (strcmp(En, "Yes") == 0 || strcmp(En, "yes") == 0 || strcmp(En, "YES") == 0) {
		if (f < Pe && i != 0) {
			//If Pe is high, add pe to ponding
			ponding = ponding + (Pe - f)*timestep;
		}
		else if (f > Pe && ponding > 0) {
			//If Pe is low, substract f from ponding
			ponding = ponding + (Pe - f)*timestep;
			if (ponding < 0) {
				//If Ponding is than 0, ponding = 0 is high, add pe to ponding
				ponding = 0;
			}
		}
		else {
			ponding = 0;
		}
	}
	else if (strcmp(En, "No") == 0 || strcmp(En, "no") == 0 || strcmp(En, "NO") == 0) {
		ponding = 0;
		Runoff = Pe - f;
	}
	else {
		cout << "ERROR: Wrong spelling of Yes or No in class ponding" << endl; //Er-
ror message returned
		exit(EXIT_FAILURE); //Exit program
	}
}
float Ponding::getPonding() {
	return ponding;
}
float Ponding::getRunoff() {
	return Runoff;
}
```

soilMoisture

soilMoisture.h

```
#ifndef RUNGEKUTTA_H
#define RUNGEKUTTA_H
class soilMoisture{
public:
	float setO0(float Z, float PeR, float O0_, float OR0L);
	float getO0();
	float getdO();
	float getk1();
	float getk2();
	float getk3();
	float getk4();
private:
	float dO;
	float O0;
	float k1;
	float k2;
	float k3;
	float k4;
};
#endif // RUNGEKUTTA_H
```

soilMoisture.cpp

```
#include "soilMoisture.h"
#include <iostream>
#include <cmath>
#include "Parameters.h"
using namespace std;
Parameters  Par;
float soilMoisture::setO0(float Z, float PeR, float O0_, float OR0L) {
	//returns parameters from the class "parameters"
	float Yb = Par.getYb();
	float Ks = Par.getKs();
	float Os = Par.getOs();
	float lambda = Par.getlambda();
	float Or = Par.getOr();
	float timestep = Par.getTimestep();
	//sets the correction factor p
	float p;
	if (PeR > 0) {
		p = 1;
	}
	else {
		p = 1.7;
	}
	//RK4 calculations k1-k4
	k1 = timestep*((1 / Z)*(PeR - Ks*(pow((O0_ - Or) / (Os - Or), 3 + (2 / lambda))) -
(p*Ks*((Yb / lambda)*(pow((O0_ - Or) / (Os - Or), (3 + (1 / lambda))) - pow(OR0L, (3 + (1
/ lambda)))) / (3 + (1 / lambda))) / Z)));
	k2 = timestep*((1 / Z)*(PeR - Ks*(pow(((O0_ + k1*0.5) - Or) / (Os - Or), 3 + (2 /
lambda))) - (p*Ks*((Yb / lambda)*(pow(((O0_ + k1*0.5) - Or) / (Os - Or), (3 + (1 /
lambda))) - pow(OR0L, (3 + (1 / lambda)))) / (3 + (1 / lambda))) / Z)));
	k3 = timestep*((1 / Z)*(PeR - Ks*(pow(((O0_ + k2*0.5) - Or) / (Os - Or), 3 + (2 /
lambda))) - (p*Ks*((Yb / lambda)*(pow(((O0_ + k2*0.5) - Or) / (Os - Or), (3 + (1 /
lambda))) - pow(OR0L, (3 + (1 / lambda)))) / (3 + (1 / lambda))) / Z)));
	k4 = timestep*((1 / Z)*(PeR - Ks*(pow(((O0_ + k3) - Or) / (Os - Or), 3 + (2 /
lambda))) - (p*Ks*((Yb / lambda)*(pow(((O0_ + k3) - Or) / (Os - Or), (3 + (1 / lambda))) -
pow(OR0L, (3 + (1 / lambda)))) / (3 + (1 / lambda))) / Z)));
	O0 = O0_ + (k1 + 2 * k2 + 2 * k3 + k4) / 6; //adds delta O0 to O0
}
float soilMoisture::getO0() {
	return O0;
}
```

Appendix C: Changes in SHyFT files

File	Lines	Type of change
Core\serialization\core_serialization.cpp\shyft	13	
	240	
	275-285	states
	336	
	410	
	469	
Core\method-stack\hbv_stack.h	6	
	33	
	48	
	49	
	60	
	93-101	Parameters
	130-138	Parameters
	170-178	Parameters
	198	
	201	
	204	
	206	
	220	
	226	
	322	
	351	
	353	
Core\method-stack\hbv_stack_cell_model.h	35-36	Response
	52-53	Response
	69-70	Response
	92-93	Response
	160-167	States
	184-191	States
	206-212	States
	225-232	States
Test\method-steck\hbv-stack-test.cpp	18	
	69	
	76	States
	88	
	89	

Appendix D: SHyFT

Hbv_infiltration_test.cpp

```
#include "fstream.h"
#include "test_pch.h"
#include "core/hbv_infiltration.h"


using namespace shyft::core;
namespace shyft {
	namespace core {
		namespace hbv_infiltration2 {
			template<class P>
			struct calculator {
				P param;
				calculator(const P& p) :param(p) {}

				double zkn(double o0,double o0l,double or0,double or0l,double 
k0,double k0l,double z, double dt){
					double G;
					if (o0 < param.os) {G = param.yb*(pow(or0, 3 + 1 / 
param.lambda) - pow(or0l, 3 + 1 / param.lambda)) / (3 * param.lambda + 1);}
					else {G = param.yb*(3 * param.lambda + 2 - pow(or0l, 3 +
1 / param.lambda)) / (3 * param.lambda + 1);}
					double k1 = dt*((k0 - k0l) / (o0 - o0l))*((G / z) + 1);
					double k2 = dt*((k0 - k0l) / (o0 - o0l))*((G / (z + 
k1*0.5)) + 1);
					double k3 = dt*((k0 - k0l) / (o0 - o0l))*((G / (z + 
k2*0.5)) + 1);
					double k4 = dt*((k0 - k0l) / (o0 - o0l))*((G / (z + k3))
+ 1);
					double zkn = z + (k1 + 2 * k2 + 2 * k3 + k4) / 6;
					return zkn;
				}
				double zk1(double o0, double o0l, double or0, double or0l, 
double k0, double z, double dt) {
					double G;
					if (o0 < param.os) { G = param.yb*(pow(or0, 3 + 1 / 
param.lambda) - pow(or0l, 3 + 1 / param.lambda)) / (3 * param.lambda + 1); }
					else { G = param.yb*(3 * param.lambda + 2 - pow(or0l, 3 
+ 1 / param.lambda)) / (3 * param.lambda + 1); }
					double k1 = dt*((k0 / (o0 - o0l)) + (param.ks*G / ((o0 -
o0l)*z)));

					double k2 = dt*((k0 / (o0 - o0l)) + (param.ks*G / ((o0 -
o0l)*(z + k1*0.5))));

					double k3 = dt*((k0 / (o0 - o0l)) + (param.ks*G / ((o0 -
o0l)*(z + k2*0.5))));

					double k4 = dt*((k0 / (o0 - o0l)) + (param.ks*G / ((o0 -
o0l)*(z + k3))));

					double zk1 = z + (k1 + 2 * k2 + 2 * k3 + k4) / 6;
					return zk1;
				}
```

```
                            double zs(double o0l, double z, double dt) {
                                   double k1 = dt*(param.ks / ((param.os - o0l))*(param.hc
/ z + 1));
                                   double k2 = dt*((param.ks / (param.os - o0l))*(param.hc
/ (z + k1*0.5) + 1));
                                   double k3 = dt*((param.ks / (param.os - o0l))*(param.hc
/ (z + k2*0.5) + 1));
                                   double k4 = dt*((param.ks / (param.os - o0l))*(param.hc
/ (z + k3) + 1));
                                   double zs = z + (k1 + 2 * k2 + 2 * k3 + k4) / 6;
                                   return zs;
                            }
                            double z0(double o0l, double or0l, double dt) {
                                   double tau = (dt*param.ks / (param.os - o0l));
                                   double G = (param.yb / param.lambda)*((3 * param.lambda
+ 2) - pow(or0l, (3 + (1 / param.lambda)))) / (3 + (1 / param.lambda)); // equation 13
GARTO
                                   double z0 = 0.5*(tau + sqrt(pow(tau, 2) + 4 * tau*G));
                                   return z0;
                            }
                            double o0(double z, double per, double o0_, double or0l, dou-
ble dt) {
                                   double p;
                                   if (per > 0) { p = 1; }
                                   else { p = 1.7; }
                                   double k1 = dt*((1 / z)*(per - param.ks*(pow((o0_ -
param.or) / (param.os - param.or), 3 + (2 / param.lambda))) - (p*param.ks*((param.yb /
param.lambda)*(pow((o0_ - param.or) / (param.os - param.or), (3 + (1 / param.lambda))) -
pow(or0l, (3 + (1 / param.lambda)))) / (3 + (1 / param.lambda))) / z)));
                                   double k2 = dt*((1 / z)*(per - param.ks*(pow(((o0_ +
k1*0.5) - param.or) / (param.os - param.or), 3 + (2 / param.lambda))) -
(p*param.ks*((param.yb / param.lambda)*(pow(((o0_ + k1*0.5) - param.or) / (param.os -
param.or), (3 + (1 / param.lambda))) - pow(or0l, (3 + (1 / param.lambda)))) / (3 + (1 /
param.lambda))) / z)));
                                   double k3 = dt*((1 / z)*(per - param.ks*(pow(((o0_ +
k2*0.5) - param.or) / (param.os - param.or), 3 + (2 / param.lambda))) -
(p*param.ks*((param.yb / param.lambda)*(pow(((o0_ + k2*0.5) - param.or) / (param.os -
param.or), (3 + (1 / param.lambda))) - pow(or0l, (3 + (1 / param.lambda)))) / (3 + (1 /
param.lambda))) / z)));
                                   double k4 = dt*((1 / z)*(per - param.ks*(pow(((o0_ + k3)
- param.or) / (param.os - param.or), 3 + (2 / param.lambda))) - (p*param.ks*((param.yb /
param.lambda)*(pow(((o0_ + k3) - param.or) / (param.os - param.or), (3 + (1 /
param.lambda))) - pow(or0l, (3 + (1 / param.lambda)))) / (3 + (1 / param.lambda))) / z)));
                                   double o0 = o0_ + (k1 + 2 * k2 + 2 * k3 + k4) / 6;
                                   return o0;
                            }
                            template <class R, class S>
                            void step(S& s, R& r, shyft::core::utctime t0,
shyft::core::utctime t1, double snow_out) {
                                   double Fp;
                                   double tp;
                                   double t = 0;
                                   double f_test;
                                   double F_test;
                                   double F_test1;
                                   double F_pot = 0;
                                   double pea = 0;
                                   double dt = 0.5/60;
                                   double F_test2;
                                   s.pe = snow_out/dt; //mm/h
                                   if (s.pe > param.ks && s.pel < param.ks && s.o0[s.k] <
param.os) {

                                          s.k++;
                                          F_test2 = param.ks * dt;
                                          s.delta_time = t0;
                                          s.delta_F[s.k] = r.F;
                                   }
                                   double pek[4] = { 0.0 };
                                   double vzk[4] = { 0.0 };
                                   double per = s.pe;
                                   for (int j = 1; j < s.k; j++) {
                                          vzk[j] = s.z[j] * (s.o0[j] - s.o0[j - 1]);
                                          s.z[j] = zkn(s.o0[j], s.o0[j - 1], s.or0[j],
s.or0[j - 1], s.k0[j], s.k0[j - 1], s.z[j], dt);
                                          pek[j] = (s.z[j] * (s.o0[j] - s.o0[j - 1]) -
vzk[j]);

                                          per = per - pek[j];
                                   }
                                   F_test1 = r.F - s.delta_F[s.k] + snow_out;
                                   s.f_test = param.ks*((param.hc*(param.os - s.o0[s.k -
1])) / F_test1 + 1);

                                   pea = pea + s.pe;

                                   Fp = ((param.hc*param.ks*(param.os - s.o0[s.k -1])) /
(s.pe - param.ks));

                                   tp = (Fp / pea)*3600;
                                   F_test = r.F - s.delta_F[s.k];
                                   if (s.pe  > 0 && s.pe  < param.ks) {
                                          r.F = r.F + snow_out;
                                          s.f = s.pe;
                                   }else{
                                          r.F = r.F + s.f*dt;
                                          s.f = param.ks*((param.hc*(param.os - s.o0[s.k -
1])) / (r.F - s.delta_F[s.k]) + 1);
                                          s.ponding = s.ponding + snow_out - s.f * dt;
                                   }
```

```
                                }else if (s.ponding > 0) {
                                        r.F = r.F + s.f*dt;
                                        s.f = param.ks*((param.hc*(param.os - s.o0[s.k - 
1])) / (r.F - s.delta_F[s.k]) + 1);
                                        s.ponding = s.ponding - s.f * dt;
                                        if (s.ponding < 0){ s.ponding = 0; }
                                }else {
                                        s.f = 0;
                                }
                                //Main if-sentences

        //New..........................................................................
.........
                                if (s.pe  > param.ks && s.pel < param.ks && s.o0[s.k] < 
param.os) {
                                        s.z[s.k] = z0(s.o0[s.k - 1], s.or0[s.k - 1], dt);
                                        if (s.pe  > s.f_test) {
                                                s.o0[s.k] = param.os;
                                        }
                                        else {
                                                s.o0[s.k] = o0(s.z[s.k], per, s.o0[s.k - 
1], s.or0[s.k - 1], dt);
                                        }
                                }
                                else {

        //C............................................................................
.......
                                        if (s.pe >= s.f_test) {
                                                s.z[s.k] = zs(s.o0[s.k - 1], s.z[s.k], 
dt);

                                                s.o0[s.k] = param.os;
                                        }

        //B............................................................................
.......
                                        double o0_test = s.o0[s.k];
                                        double z_test = s.z[s.k];
                                        if (s.pe  < s.f_test && s.pe >= param.ks) {
                                                double F_test = r.F - s.delta_F[s.k] - 
s.f*dt;
                                                while (F_test < r.F - s.delta_F[s.k]) {
                                                        o0_test = o0_test + 0.0001;
                                                        z_test = zk1(s.o0[s.k], s.o0[s.k - 
1], s.or0[s.k], s.or0[s.k - 1], s.k0[s.k], s.z[s.k], dt);
                                                        F_test = z_test*(o0_test - s.o0[s.k 
- 1]);
                                                }
                                                s.z[s.k] = z_test;
                                                s.o0[s.k] = o0_test;
                                        }
```

```
      //A.................................................................
                                            if (s.pe  < param.ks) {
                                                  if (s.ponding > 0) {
                                                        s.z[s.k + 2] = zs(s.o0[s.k - 1], 
s.z[s.k], dt);

                                                        s.o0[s.k] = param.os;
                                                  }
                                                  else {
                                                        float F_test = r.F - s.delta_F[s.k] 
- 0.001;

                                                        while (F_test < r.F - 
s.delta_F[s.k]) {

                                                              z_test = z_test  + 0.001;
                                                              o0_test = o0(s.z[s.k], per, 
s.o0[s.k], s.or0[s.k - 1], dt);

                                                              F_test = z_test*(o0_test - 
s.o0[s.k - 1]);
                                                        }
                                                        s.z[s.k] = z_test ;
                                                        s.o0[s.k] = o0_test;
                                                  }
                                            }
//.........merges to wetting fronts if they reach the same level........................
                                            if (s.z[s.k] > s.z[s.k - 1] && s.k > 0) {
                                                  s.z[s.k - 1] = ((s.o0[s.k] - s.o0[s.k - 
1])*s.z[s.k] + (s.o0[s.k - 1] - s.o0[s.k - 2])*s.z[s.k - 1]) / (s.o0[s.k] - s.o0[s.k - 
2]);
                                                  s.o0[s.k - 1] = s.o0[s.k];
                                                  s.z[s.k] = 0, s.o0[s.k] = 0, s.k0[s.k] = 
0, s.or0[s.k] = 0;
                                                  s.k--;
                                            }
                                      }
                                      s.or0[s.k] = (s.o0[s.k] - param.or) / (param.os - 
param.or);
                                      s.k0[s.k] = param.ks*pow(s.or0[s.k], 3 + 2 / 
param.lambda);
                                      s.pel = s.pe;
                                }
                          };
                    }
            }
};
```